uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería en Tecnologías de Telecomunicación
2016-2017

Trabajo Fin de Grado

SISTEMA DE SEGMENTACIÓN AUTOMÁTICA DE SECUENCIAS DE VÍDEO PARA TRACKING DE PUBLICIDAD

Adriano Augusto Osses Rodríguez

Tutor

Julio Villena Román

Leganés. Octubre 2017

RESUMEN

Se ha estudiado, diseñado e implementado un sistema para realizar un seguimiento o *tracking* de publicidad. La aplicación analiza secuencias de vídeos de espacios publicitarios en cadenas de televisión. El sistema primero realiza una segmentación del vídeo, identificando el tiempo cuando se empieza a emitir publicidad y cada anuncio, y después identificando la empresa anunciante por sus logotipos. Para ello se utilizará Visión Artificial. El código está programado en **Python**, utiliza la librería **OpenCV** y otras librerías auxiliares relacionadas con el manejo de matrices.

El sistema tiene como entradas un vídeo el cual se quiera analizar, y una carpeta con las imágenes de los logotipos que identifiquen a las empresas que se quieran buscar. Como salida, el programa indica el intervalo de tiempo exacto en el cual aparece dicho anuncio, y el nombre el archivo de imagen encontrado. La búsqueda de imágenes se hará mediante *Feature Matching.* Se utilizan y se describen los algoritmos más importantes.

Los temas principales de este proyecto son la Visión Artificial y la Publicidad. Se estudia su relación entre ellos y con el programa creado; las aplicaciones; cómo influyen en la sociedad y como han evolucionado en la Era de Información.

EXTENDED ABSTRACT

At present, human beings are living the Information Age. We receive more amount of information and faster than ever. Advertising and marketing exists thousands of years ago. Sellers and marketing experts study how improve profits of companies every day. They use new communication channels to transmit all type of information in few seconds.

Malcolm Gladwell explains in his book, *The Tipping Point* [Gladwell00]*: "Media Dynamics Inc. estimates that the average American is exposed to 254 different commercial messages in a day".* We need new tools to analyze this information flow, as Data Analysis and Big Data.

The human brain is unknown organ now yet. Daniel Kahneman address this in his book *Thinking Fast and Slow* [Kahneman11]. The psychologists Keith Stanovich and Richard West originally proposed two terms to refer to two systems in the mind: System 1 and System 2. "System 1 operates automatically and quickly, with little or no effort and no sense of voluntary control". On other hand, "System 2 allocates attention to the effortful mental activities that demand it, including complex computations".

When someone is watching TV programs, commercials, advertisement, this observer is received more information than he believes. Frequently, after of large TV session, this spectator doesn't remember all he saw seconds ago. Marketing experts know this. They employ multiple resources to impact in System 2. Catchy songs, showy logotypes, repetition, are some techniques used.

Our specie has gone through an evolution process of millions of years. Our brain has learnt to identify external threat thousandths of a second. Now, programmers are trying that machines can see and interpret images.

Gary Bradski and Adrian Kaehler define Computer Vision in his book, *Learning OpenCV* [Bradski08]. "Computer vision in the transformation of data from a still or video camera into either a decision or a new representation. All such transformations are done for achieving some particular goal". In this chapter, the authors describe how works the human vision. We have hardware sensors, the eyes, which mechanically control lighting via the iris and tune the reception on the surfaces of the retina. The human brain divides the vision signal into many channels with different kinds of information in the brain. The Computer Vision works differently.

Machine receives a video, a frames sequence. This information is analyzed as a grid of numbers. Each pixel contains information of color and intensity, but has a rather large noise component, and so by itself give little information.

In this project, it is described, programmed and analyzed some algorithms of Computer Vision. It created a system programmed in Python, with the library of Computer Vision, OpenCV.

This Bachelor Thesis is named Automatic Segmentation System of Video Sequences for Ad Tracking. The project goals are:

- **Developing an automatic system to identify brand logotypes in a video sequences** for Ad Tracking. The video sequences can be part of a TV show or event, or commercial time. In this part of video, screen shows different logos during emission.

- **The system receives an input video file, and prints a text file with a record for each logo, and prints results in console.** The program shows in console when a commercial appear in video, in timeline. The records include three columns: first column for number

of matches in an image; second column for number of correct matches; and third for time interval.

Mark Lutz define **Python** as "a powerful multiparadigm computer programming language, optimized for programmer productivity, code readability, and software quality". In his book, *Learning Python* [Lutz13], this author classifies and describe the advantages of Python.

- **Software quality.** Python is more readable than traditional scripting languages. Hence, it's more reusable and maintainable. Python support object-oriented and function programming.

- **Developer productivity.** Python code takes up less space than languages such as C++ or Java code. Python programs run immediately. It's not necessary compile or link steps.

- **Program portability.** Python programs run on a major of computers platforms. These platforms can be Operative Systems as Linux or Windows. Python offers options for database access programs or web-based systems

- **Support libraries.** Python includes a self-library named standard library, that support**s** different functionalities as array of application-level programming task or network scripting. Python can be extended with other libraries and support software. For example, in this project, it's used OpenCV.

- **Component integration.** Python has integration mechanisms to communicate with other parts of and application.

Some of applications that use Python [Lutz13] are:

- **Google**, in web search system.
- **YouTube**, in its video sharing service.
- The **BitTorren**t peer-to-peer file sharing system.
- **Pixar** in the production of animated movies.
- **ESRI** in its GIS mapping products.
- **Google's App Engine** web development framework
- The **NSA**, in cryptography and intelligence analysis.
- **Netflix**, in software infrastructures.
- **JPMorgan Chase**, in financial market forecasting.
- **Nasa**, for scientific programming task.

In this project, it's worked with **OpenCV** (Open Source Computer Vision Library). Originally, this library was developed by Intel. After, it was supported by Willow Garage, and now by Itseez[1]. OpenCV is released under a BSD license.

OpenCV's Official Site[2] enumerates some applications of these algorithms.

- Detect and recognize faces.
- Identify objects.
- Classify human actions in videos.
- Track camera movements.
- Track moving objects.

[1] Wikipedia. OpenCV. https://en.wikipedia.org/wiki/OpenCV
[2] OpenCV site. http://opencv.org/

- Extract 3D models of objects.
- Produce 3D point clouds from stereo cameras.
- Stitch images together to produce a high-resolution image.
- Find similar images from an image database.
- Remove red eyes from images taken using flash.
- Follow eye movements.

It supports Windows Linux, Mac Os iOs and Android. It has C++, C, Python and java interfaces. Many companies use this library: Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota. And startups as Applied Minds, VideoSurf and Zeitera. This companies use OpenCV for[3]:

- Span the range from stitching streetview images together.
- Detecting intrusions in surveillance video.
- Monitoring mine equipment in China.
- Helping robots navigate and pick up objects.
- Detect drowning accidents in swimming pools.
- Inspecting label products in factories.

In addition, other libraries used are Numpy and Matplotlib. **NumPy** is a package for scientific computing with Python. The previous version, Numeric, was created by Jim Hugunin, with other developer collaboration, in 1995. In 2005, Travis Oliphant developed Numpy[4]. In his Official Site[5], enumerate some functions: a powerful N-dimensional array object; broadcasting functions; useful linear algebra; Fourier transform, random number capabilities and more. Numpy has BSD license too.

Matplotlib is a library intended to generate graphics, useful to work with mathematical graphics. It was written by John D. Hunter and other developers in 2003[6]. Matplotlib is an object-oriented application. PyLab module contains functions to create graphics. It provides an interface MATLAB-style[7].

The sections and algorithms related with feature detection and descriptions are the following (the official documentation[8] contains explanations and tutorials):

- **Features.** The features of an image are patch, or set of pixels, easily identifiable. This features usually are corners. There are different algorithms to detect these corners. The action of finding features is called Feature Detection.

- **Harris Corners Detection.**

- **Shi-Tomasi Corner Detector.**

- **SIFT** (Scale-Invariant Feature Transform).

[3] OpenCV site. http://opencv.org/
[4] Wikipedia: NumPy. https://en.wikipedia.org/wiki/NumPy
[5] Numpy Official Site. http://www.numpy.org/
[6] Wikipedia: MatplotLib. https://en.wikipedia.org/wiki/Matplotlib
[7] Matplotlib Official Site. https://matplotlib.org/
[8] OpenCV Documentation. Docs OpenCV. http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html

- **SURF** (Speeded-Up Robust Features).
- **FAST** Algorithm for Corner Detection.
- **BRIEF** (Binary Robust Independent Elementary Features).
- **ORB** (Oriented FAST and Rotate BRIEF).
- **Feature Matching.** In this document, it's explained the Brute-Force Matcher (BFMatcher) and FLANN based Matcher (Fast Library for Approximate Nearest Neighbors).
- **Homography** to find Objects. These functions let use the algorithms RANSAC and LEAST_MEDIAN, for solved possible errors while matching.

About Socio-economic background. The code has been programmed in one laptop computer. The operative system chosen is Windows 10. In addition, it's rented a Droplet to backup and possible running in Linux. The programming environment is Visual Studio 2017. This program signals the errors in lines, and has IntelliSense technology. This Visual Studio technology can autocomplete code, and show library functions to help developers.

The activities to create this project are divided in four sections:

1. **Problem analysis and information gathering.** These activities are linked to identified the problem to solve, the main points, and current resources. The second step is search and classifier this information of different sources (books, articles, Internet, forums, etc.)
2. **Preparation of environment.** A powerful hardware is necessary to programming and running the code many times. We need programs to image, video and text edition too.
3. **Program creation.** The first step is learning Python and OpenCV tools: test code to identify an image in other image, image on video, and many images on video.
4. **Writing document.**

Concerning **budget**. The direct cost (programmer salary, laptop-computer, programs) is 9,080.14 €. The indirect cost (energy bill, Internet, office equipment) is 1,816.02. The total project cost is 10,926.16 €.

The **exploitation plan** thought for this project is linked with Ad Tracking. In the Information Age, we have vast amounts of data, every second, and we need tools to analyze this information to understand what is going on. Against this backdrop, all companies want to maximize their profits. Many of these profits are produced thanks to publicity and marketing.

In this project, it's created an Automatic System for Ad Tracking. Systems means that program receives inputs (images and video) and generate outputs (records). Automatic means that algorithm obtains the results by itself, it can "visualize" any video and search a big set of logotypes in it. This, companies can study the influence and the impact of their commercials. They can decide what features (type of advertisement, in what slot time, what frequency, etc.) are the best. Identifying strengths and weaknesses is essential in any company.

So, in this project it's used image of logotypes. These images are protected with **Intellectual Property** (IP). It's toke it, to write this document and testing the program, for educational and non-commercial purposes.

The **WIPO** (Word Intellectual Property Organization) enumerate the different types of IP rights exist linked with advertising campaign[9]. Some are:

- **"Creative content**. Written material, photographs, art, graphics music and videos. Protection: copyright".
- **"Slogans** and **sound**. Protection: copyright".
- **"Sign.** Business names, logos, product names, domain names. Protection: trademarks".
- **"Geographical indications.** Protection: laws against competition, appellations of origin" and others.
- **"Computer-generated graphic symbols, graphics user interfaces, screen displays, and web pages**. Protection: industrial design law".
- **"Website design**. Protection: copyright".

As far as **methodology**, the first step is the images and videos set. The **videos** must be clear, without reproduction errors, with the maximum smoothness of motion, to get a better detection. It's used **VLC Player** to record streaming videos. This is free software and open source cross-platform multimedia player.

To the images set, it should be noted some factors:

- The **logotype must be current**, or similar to logo showed in commercials.
- For a better detection, to use images with **same background** than logo showed in commercials is recommended.

Once the set has been chosen, we must select the best images.

- For each commercial, it's run tests with the sets. Then, the best image (the image which generate more matches) is chosen. If any image works, another set can be tested.
- OpenCV have tools (histogram equalization, of noise suppression) to improve the images.

The **system goal** is, read logos image files (JPG, PNG, GIF) and commercial video files (.TS or other), and write a text (.txt) that contains when and what logo appear in video. To run the program, the user must write this line in Console (in programa.py path):

```
python programa.py datosX video_test.ts
```

Where:

1. **python**: argument needed to run a Python code.
2. **programa.py**: Python program's name. It contains the main.
3. **datosX**: folders name that contains the logo images.
4. **video_test.ts**: file videos name to analyze.

Program operation. First, the algorithm read arguments introduced. Each logo image has an id. The program run a search function in parallel. To use threads in python, the code should contain the line to import the package concurrent.futures.

The search function receives an image id and the commercial video. A part of function obtains the image file and the name of logo from this id. To identifying the image in a video frame, program must detect the *features (corners)*. To this end, we use SIFT and SURF detectors, and FLANN and Brute Force matchers. If the number of matches exceeds a predefined threshold,

[9] WIPO Official Site. http://www.wipo.int/portal/en/index.html

RANSAC algorithm received it and selects the correct matches. If number of correct matches exceeds a second predefined threshold, the logo has been detected. Then, a detection flag is activated. Each second (or fraction of second), algorithm write the record in output text file.

At a time, the program is detecting black frames. Black frames appear in transition between commercials. When the number of pixels non-black pixels are less than a threshold, the program detects the end of a commercial, and the start of another commercial. Then, read the detection flag value. If flag is activated, a line with the interval time and logotype file name is showed in console.

Results. We "trained" the algorithm with 8-images set to choose parameters values (thresholds, and detector and matchers parameters). Next, we test the program with another 10 images set and other video with commercials. The accuracy is 90%.

Conclusions and Future jobs. The segmentation is correct. SIFT provides better results than SURF. The reasons of error in logos usually are:

- No compact logos.
- Too rounded logos.
- Very small logos
- Briefly show.
- External sources effects.

Possible improvements can be: machine learning algorithms, use Haar Cascade algorithm, use better hardware, and work with audio.

Contenido

Índice de Figuras

Índice de Tablas

1. INTRODUCCIÓN

1.1. PLANTEAMIENTO DEL PROBLEMA

Con la imagen siguiente inicia Daniel Kahneman el primer capítulo de su libro, *Pensar rápido, pensar despacio*[10].

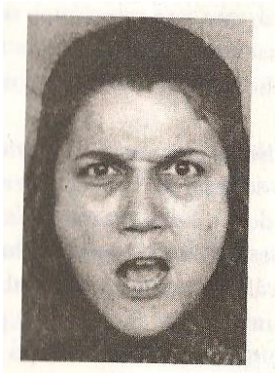


Figura 1: Rostro de una persona enfadada para entender el pensamiento intuitivo. [Kahaneman11]

Su función: hacer consciente al espectador de su modo automático, y de enseñar lo que es la visión y el pensamiento intuitivo. Al instante de ver la foto, uno *sabe*, que la mujer de la imagen está enfadada, es más, puede hacer predicciones sobre lo que pasará a continuación. Todo esto, sin ningún tipo de esfuerzo.

En el libro además se hace referencia a dos términos propuestos por los psicólogos Keith Stanovich y Richard West: el Sistema 1, que opera de manera rápida y automática, con poco esfuerzo y sin sensación de control voluntario; y el Sistema 2, que centra la atención en las actividades mentales esforzadas que lo demandan.

Este es un buen ejemplo para entender la complejidad de un cerebro un humano. Las razones evolutivas tienen sentido: aquellos seres humanos en la prehistoria que supieron identificar mejor las amenazas sobrevivieron y dejaron descendencia. Un niño pasa los primeros meses de su vida asociando las imágenes que percibe a sentimientos y conceptos. El reto actual: pretender realizar todo este proceso a partir de transistores.

Como se vio en el ejemplo anterior, las personas pueden hacerse una idea de lo que están mirando realmente. Si se les pregunta, podrán describir con todo detalle lo que ven a través de una pantalla. Pero lo cierto es que esto solo será una pequeña cantidad de información. El resto ya está en su inconsciente.

Esta es una razón por la que algunos programas tienen éxito sin que nadie pueda explicar por qué. Pero la razón existe. En este proyecto se analizará uno de los canales más representativos de este fenómeno: la televisión, y más concretamente, la publicidad.

En su libro La Clave del Éxito, Malcolm Gladwell [Gladwell00], estudia cuáles son los factores que hacen que un programa, una moda o una ideología triunfe sobre otras. Dedica un capítulo al éxito de programas infantiles emitidos por televisión. Destaca la «participación baja» al ver la televisión, y hace referencia a un experimento, en los que se divide a un grupo de niños en dos subgrupos: en el primer, los niños debían leer un texto y se les dijo que después debían contestar a una serie de preguntas; el segundo subgrupo solo debía ver un vídeo sobre el mismo tema. El

[10] Capítulo 1: Los personajes de la historia (pág. 33). [Kahaneman11]

resultado fue que los niños del primer subgrupo obtuvieron mejores puntuaciones. Es decir, muchas personas ven la televisión y al rato, el Sistema 1, el consciente, quizás no se acuerde de lo que haya visto ese día. Ni hablar de un mes después. Sin embargo, el Sistema 2 ya *sabe* que la Coca-Cola es la chispa de la vida.

En el capítulo se habla del *factor gancho,* y cómo funcionan algunas técnicas empleadas en publicidad para que un mensaje sea memorable.

La repetición es un factor importante para que la gente recuerde un producto. Entre los publicistas existe una máxima: «es necesario ver un anuncio al menos seis veces para que la gente empiece a recordarlo. Se trata de una lección muy útil para marcas como Coca-Cola o Nike, que invierten cientos de millones de dólares en publicidad y que pueden saturar a todos los medios de comunicación con su mensaje» [Gladwell00]. En este proyecto se crea, precisamente, un sistema que realizara este conteo de forma automática.

Llamar la atención del espectador no es tan trivial como puede parecer a simple vista. Cualquiera puede ver un anuncio con determinado éxito, y dar una serie interminable de razones, totalmente justificadas, del porqué de su éxito, todas a posteriori. El escenario es completamente distinto a la inversa: pretender crear un anuncio y que tenga éxito. Aquí toda la teoría inicial se viene abajo. No se puede predecir lo altamente improbable, como se explica Nicholas Nassim Taleb en *El Cisne Negro* [Taleb07].

Volviendo al libro de Gladwell [Gladwell00], los expertos el *marketing* directo conocen este problema, y tiene solución. Hacen muchas pruebas para determinar el impacto de un anuncio, y qué cosas funcionas y cuáles no. Para ello, crean distintas versiones un mismo anuncio, y las emiten en lugares diferentes. Después, analizan la respuesta de cada versión en cada lugar. Sin embargo, hay un elemento que con seguridad se repetirá en todos los anuncios: la marca.

En los anuncios de televisión, ésta vendrá representada principalmente por una imagen de un logotipo, y adicionalmente, para reforzar su recuerdo, se añade un lema y una música. Ray Kroc, fundador de McDonalds, demostró que la marca puede convertir un pequeño puesto de comida, en una empresa multimillonaria. En la película *The Founder* (2016), se explica esta historia, y se enfatiza la obsesión del protagonista por el nombre de la marca, el logotipo de los arcos dorados, y la idea que se quiere asociar a la marca: la rapidez, y un lugar de reunión agradable.

En este proyecto se identificará, a través de algoritmos de Visión Artificial, qué logotipos aparecen en determinados fotogramas, y se esa forma identificar el anuncio para realizar el conteo. Se podrá saber, de forma automática, en qué momentos del día aparece un anuncio, cuál es su duración, y cuál es su frecuencia de aparición.

Es este análisis objetivo lo que hará optimizar una empresa y generar más beneficios. En el caso de los publicistas, éstos se dividen en dos categorías. Los publicistas convencionales actúan basándose en ideas tradicionales, creen que para que un anuncio tenga éxito debe «ser divertido, el diseño debe ser ostentoso, y mejor si aparece la imagen de un famoso». Por otro lado, están los expertos en marketing directo que se basan en medidas objetivas para medir la eficacia de su idea. Estas medidas pueden ser «la cantidad de cupones-respuesta recibidos o la cantidad de personas que llaman a los números 900»Ô [Gladwell00].

El segundo método se acerca más a la idea de cómo funciona la propia naturaleza: generar acciones aleatorias y seleccionar la más apta. Los motivos que alguien puede dar por los que un anuncio ha tenido éxito pasan a un segundo plano. Quizás pueden ser un punto de partida, pero al final lo que va a interesar son los resultados.

La publicidad funciona. Y la publicidad en televisión funciona muy bien. De momento. La empresa Media Dynamics «calcula que el estadounidense medio está expuesto a 254 mensajes publicitarios diferentes cada día» [Gladwell00]. En España, consumo de televisión se refleja en la siguiente gráfica, perteneciente a un artículo del periódico El País[11]:

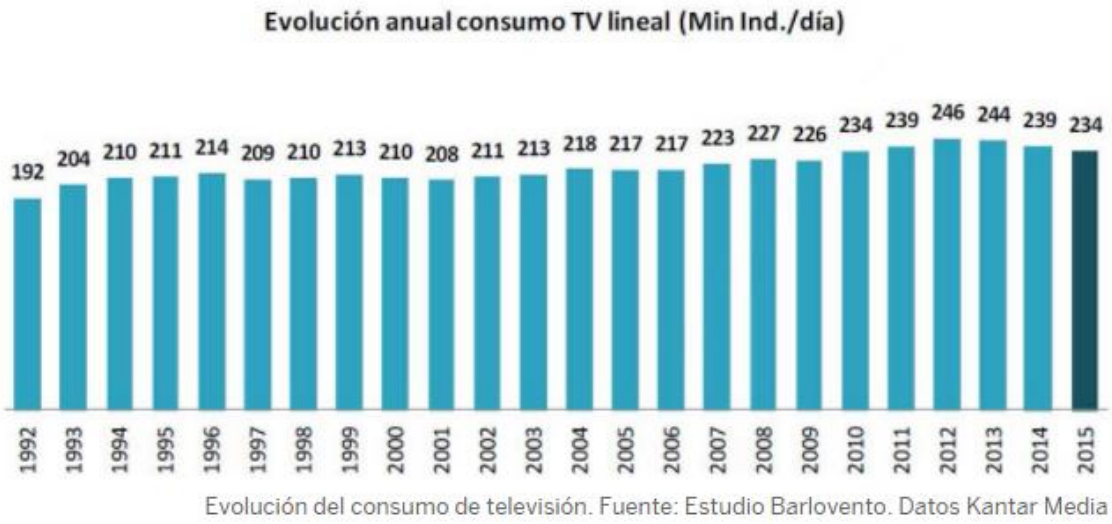


Figura 2: Evolución del consumo de televisión en España.

Se observa un reciente decremento desde el 2012, posiblemente debido a la proliferación de Internet. Lo que no hace que la publicidad y los anuncios originalmente emitidos por televisión, desciendan, más bien al contrario.

En este nuevo escenario rebosante de información por todas partes, son necesarias herramientas de Análisis de Datos para poder analizar de forma rápida y global, cuáles son estos datos, cómo se clasifican, a qué sector de la población están dirigidos, qué efectos producen, etc. Y es imperativo que gran parte de este proceso sea automatizado.

1.2. OBJETIVOS DEL PROYECTO

1. **Desarrollar un sistema automático para identificar logotipos de marcas en secuencias de vídeo** para realizar tracking de publicidad, empleando las herramientas que ofrece la librería OpenCV.

   Las secuencias pueden ser de dos grupos. En el primero, las imágenes corresponderán con el contenido emitido por la cadena, ya sea un programa, un evento deportivo o una película. Estas imágenes, por tanto, contarán con una mosca, que será el logotipo de la cadena, situado por lo general una de las cuatro esquinas del recuadro que contiene la imagen. El segundo grupo corresponderá al espacio publicitario. Las secuencias mostrarán, en algún momento, una imagen representativa de la marca. Por lo general, en primer plano, y con un fondo determinado.

[11] (18/05/2017). Los españoles recortan el tiempo que ven la televisión al día: 243 minutos. Madrid. El País. https://economia.elpais.com/economia/2016/05/18/actualidad/1463571226_893173.html

Para ello se van a utilizar algunas de las librerías que ofrece OpenCV, y se va a analizar su rendimiento y el nivel de acierto.

1. **El sistema deberá recibir como entrada un archivo de vídeo, y deberá imprimir un registro**. Este registro contendrá el instante de tiempo, si en ese momento se está emitiendo contenido de la cadena o publicidad, y en el segundo caso, indicar qué se está anunciando.

1.3. CONTENIDO DE LA MEMORIA

El presente documento cuenta con seis capítulos, bibliografía y un anexo.

En el **primer capítulo** se da una **introducción** de la visión artificial, como se relaciona con el cerebro humano y se describe cual es el planteamiento del problema. Se habla de los cambios que ha tenido la publicidad, el aumento de la información, y explica la necesidad de este tipo de proyectos. Se han descrito los objetivos de este trabajo, qué entradas entra en el sistema, y qué salidas proporciona.

El **segundo capítulo** es el **Estado del Arte**. En él se da una breve introducción al funcionamiento de la visión artificial, qué aplicaciones tiene, cuáles son las dificultades que entraña y qué soluciones se han encontrado. En las siguientes secciones del capítulo se realiza una explicación del lenguaje utilizado en este proyecto, Python, y de las librerías utilizadas, en especial Open CV 3. Se explica cuáles son sus ventajas e inconvenientes, y sus aplicaciones. A continuación, describe el funcionamiento de los algoritmos más relevantes para este proyecto.

En el **tercer capítulo**, se analizará el Entorno Socio-Económico y el Marco Regulador. En el **Entrono Socio-Económico** se habla de cuáles son los materiales *hardware* y *software* empleados. Se justifica el entorno: qué SO, entorno de programación y lenguaje se ha escogido y por qué. Se dedica una sección al Presupuesto, en el que se detalla cual ha sido el presupuesto del proyecto, indicando en un diagrama de Gantt las actividades realizadas, los costes directos e indirectos, y un plan de explotación en el que se detalla cual es el impacto del proyecto.

En el **Marco Regulador** se expone una advertencia relacionada con este mismo trabajo. También incluye una sección sobre los tipos de derechos que tiene la Propiedad Intelectual en la publicidad.

El **cuarto capítulo** corresponde a la **Metodología Empleada**. Primero, se muestra cómo y con qué se han obtenido los recursos de entrada (cortes de vídeo, e imágenes de logotipo), y qué características deben cumplir. Seguidamente, se lista la biblioteca de estos recursos con la que se va a trabajar, con sus características.

A continuación, se detallan todos los pasos que se han seguido para la creación del proyecto. En un principio se expone cual es el propósito del programa. Después, se enseña cómo funciona el código con diagramas de flujo y una explicación. Se presenta la organización del programa y se explica cómo debe ordenarse en un directorio.

En el **quinto capítulo** se analizan los **Resultados Obtenidos**. Se evalúan las distintas pruebas realizadas: se mide el nivel de acierto, se muestran gráficas con los resultados y la matriz de confusión. Se establecen los parámetros del programa para después realizar una prueba de detección en una secuencia más larga de anuncios. Se mide la probabilidad de acierto y el tiempo de ejecución.

En el **sexto capítulo** se exponen las **Conclusiones** y se proponen **Trabajos Futuros**. En él se analizan los resultados obtenidos en las pruebas, y se separan las ventajas y convenientes de las herramientas y recursos utilizados. Se proponen trabajos futuros y posibles mejoras del algoritmo. Se estudia de qué forma optimizarían el código, y cómo se obtendrían mejores resultados.

En la **bibliografía** consultada, se incluyen libros y artículos. Seguidamente, se listan algunos recursos de internet que han sido utilizados, pero no se han citado a lo largo del documento.

Por último, se incluye un **Anexo** con el manual del usuario para preparar el programa, ejecutarlo, y visualizar los resultados.

2. ESTADO DEL ARTE

2.1. VISIÓN ARTIFICIAL

Gary Bradski y Adrian Kaehler definen en su libro (*Learning OpenCV*) [Bradski08] la **Visión Artificial** (*Computer Vision*) como «la tranformación de los datos de que nos proporciona una fotografía o un vídeo, en una decisión o nueva representación. Todas estas transformaciones se hacen con un objetivo particular».

En el libro se exponen ejemplos de datos de entrada y decisión. Los datos pueden ser los proporcionados por una cámara colocada en un coche, o un medidor de distancia láser. La decisión un algoritmo puede ser si aparece una persona en un vídeo, o que hay cierto número de células tumorales en una diapositiva.

En la misma fuente [Bradski08], se describe como un ser humano ve y reconoce algo. El cerebro humano divide una señal, percibida a través de la vista, en muchos canales que transmiten diferentes tipos de información al cerebro. Éste tiene un sistema de atención que identifica las partes importantes de una imagen para examinarlas. Estas tareas dependen unas de otras, y existe una retroalimentación en el flujo visual que aún es poco entendida. Existe una gran cantidad de información de entrada, generada a partir de sensores de control muscular y de los sentidos, que permiten al cerebro basarse en asociaciones cruzadas que se crean a lo largo de años de vida. Un ejemplo de estos sensores, que serán el *hardware*, son los ojos. Controlan la iluminación mecánicamente a través del iris, y sintonizan la recepción en la superficie de la retina. La siguiente imagen pertenece al capítulo introductorio del libro *Computer Visión*. [Bradski08]:

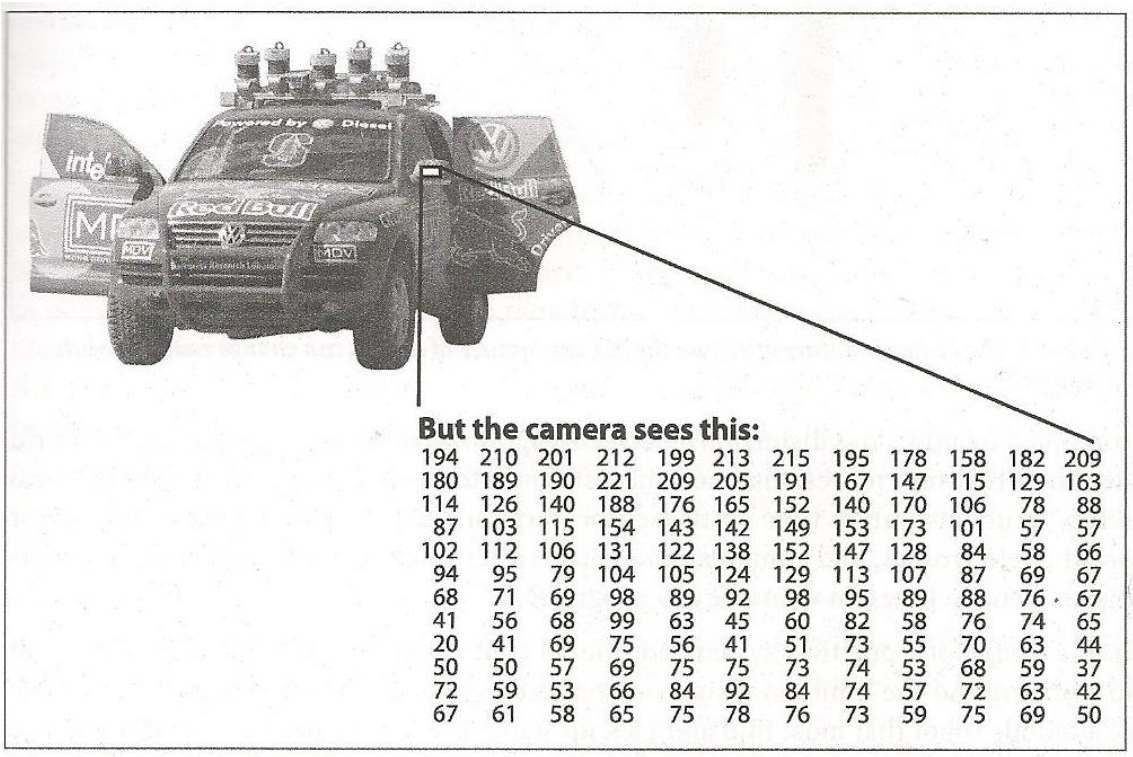


Figura 3: Representación de una imagen vista por el cerebro humano, y su equivalente para una máquina.

En la imagen superior se representa lo que el ojo humano ve, la fotografía de un coche en escala de grises, y concretamente, identifica un retrovisor. En un sistema de visión artificial, un ordenador recibe una matriz de números. En un primer momento, no hay control automático de enfoque y apertura, ni asociaciones cruzadas formadas con los años de experiencia, como en el caso de los humanos. Los números de esta matriz tienen una componente de ruido elevado, y por sí mismo, nos da poca información. La tarea, en este ejemplo, consistirá en *convertir* estos datos en un retrovisor.

Es una tarea complicada. Con esos datos, una misma imagen 2D podría representar una combinación infinita de escenas en 3D, incluso si fuesen datos perfectos. El problema así está mal planteado.

Los datos extraídos de la imagen anterior pueden estar alterados debido al ruido y otras distorsiones. Las diferentes causas pueden ser[12]:

- **Ruidos técnicos**: *salt and pepper,* ruido uniforme, ruido gaussiano.
- **Interferencias debidas al contexto**: punto de vista, oclusión, escala, deformación, fondo desordenado, variaciones dentro de una misma clase.
- **Otras variaciones del entorno**: tiempo, iluminación, reflejos, movimiento.
- **Imperfecciones en la lente** o en la **configuración mecánica**.
- **Compresión de la imagen**.

Figura 4: cambios que se producen cuando cambia la iluminación en un rostro.[13]

Para crear un diseño de un sistema práctico, se debe tener en cuenta el contexto. Para reconocer un objeto en una imagen, se puede almacenar un conjunto de imágenes en una base de datos. Las imágenes recopiladas, las fotos realizadas por la gente, puede contener información adicional implícita.

Para solucionar los problemas en los datos por el ruido se usan métodos estadísticos. Para detectar un borde -una parte característica de una imagen (se tratará en las secciones siguientes)-, se deben mirar las estadísticas en una región, y no solo basarnos en un píxel y sus vecinos inmediatos (píxeles, o regiones de la imagen cercanas). Otra opción es tener en cuenta el ruido producido por el aparato físico. Se pueden construir modelos aprendidos de los datos disponibles con el que corregir las distorsiones [Bradski08].

[12] Visión Artificial. Wikipedia en español. https://es.wikipedia.org/wiki/Visi%C3%B3n_artificial
[13] https://www.cs.princeton.edu/courses/archive/spring07/cos424/lectures/li-guest-lecture.pdf

La visión artificial tiene diversas aplicaciones[14]:

- **Sistemas de visión industrial**: e.g. maquinas que inspeccionan botellas en una línea de producción
- **Inteligencia Artificial**: el aprendizaje automático (*machine learning*) se basa en el modo en el que las máquinas pueden solucionar problemas *aprendiendo.* Andrew Ng explica en un curso de *Machine Learning*[15]*,* un ejemplo de esta aplicación. El experimento realizado por Dean Pomerleau muestra como una máquina puede aprender a conducir un coche, utilizando redes neuronales.

Figura 5: Fotograma extraído del vídeo que muestra el aprendizaje automático de una máquina para conducir un coche a partir de la visión de una carretera y los movimientos del conductor. El vídeo pertenece al curso de Coursera «Machine Learning», semana 5: Neural Networks: Learning.

En el cuadro inferior derecha se muestra un mapa de la ruta que sigue el coche. En el cuadro inferior izquierda se representa una serie de píxeles, en una resolución determinada y en escala de grises, que representa lo que la maquina está *viendo.* En el cuadro superior izquierda hay dos barras negras en horizontal. Estas barras representan el movimiento del volante. En la superior, un poco a la izquierda de la mitad, hay una zona de punto blanco concentrados, es movimiento que realiza el conductor humano. Debajo, también existe una zona de puntos blancos o grises, pero más dispersos, y corresponden a la acción de que toma la máquina. A medida que pasa el tiempo, la maquina aprende a tomar decisiones a partir de la imagen inferior y los datos reales del conductor.

- **Inspección automática**
- **Sistemas de identificación.**
- **Procesos de control.**
- **Detección**: vigilancia o conteo de personas. Reconocimiento facial.
- **Interacción ordenador-humano**.
- **Modelado de objetos**.
- **Navegación**.

[14]Computer Visión. Wikipedia en inglés. https://en.wikipedia.org/wiki/Computer_vision
[15] Curso de Machine Learning. Coursera. https://www.coursera.org/learn/machine-learning

- **Organización de información**

2.2. LENGUAJES DE PROGRAMACIÓN EMPLEADOS

2.2.1. PYTHON

Figura 6: Logotipo actual de Python.

Python es un potente lenguaje de programación multiparadigma, optimizado para la productividad del programador, la legibilidad del código y la calidad del software. [Lutz13]
Hay aproximadamente un millón de usuarios de Python por el momento. Mark Lutz divide sus características en los siguientes grupos.

La calidad del software, la legibilidad y la coherencia lo diferencia de otras herramientas de secuencias de comando. El código Python está diseñado para ser legible y por lo tanto reutilizable y sostenible. Su uniformidad hace que sea fácil de entender. Tiene soporte para los mecanismos de reutilización de software avanzados, como la programación orientada a objetos (POO).

Productividad del desarrollo. Lo que ocupa un código de Python es un tercio o hasta un quinto del tamaño de un código equivalente en C++ o Java. Lo que se traduce en menos tiempo de escritura y depuración. Python no necesita compilación ni los pasos requeridos por otras herramientas, se ejecuta inmediatamente aumentando la velocidad de programación. La mayoría de los programas Python se ejecutan son variaciones en las principales plataformas. Ofrece múltiples opciones para codificar interfaces gráficas, programas de acceso a bases de datos, sistemas basados en web, y más.

Python contiene una gran colección de funciones pre-programadas y portables, conocida como ***standard library***. La cual soporta tareas de programación a nivel de aplicación: desde la correlación de patrones de texto, hasta la creación de secuencias de comandos en red. Puede ampliarse tanto con librerías locales, como con una gran colección de software de terceros. Este ofrece herramientas para la creación de sitios web, programación numérica, acceso a puertos serie, desarrollo de juego, y más.

Los scripts de Python se pueden comunicar fácilmente con otras partes de una aplicación, utilizando una variedad de mecanismos de integración. Esto permite que Python sea utilizado como una **herramienta de personalización y extensión de productos**. El código Python puede invocar bibliotecas C y C++; puede ser llamado desde programas C y C++, puede integrarse con componentes Java y .NET; puede comunicarse sobre frameworks como COM y Silverlight; puede interactuar con dispositivos a través de puertos serie; y puede interactuar sobre redes con interfaces como SOAP, XML-RPC y CORBA.

Lutz enumera algunos de los usos de Python por empresas en la actualidad [Lutz13]:

- Sistemas de búsquedas web de **Google**.
- **YouTube** está escrito en Python en gran parte.
- El servicio de almacenamiento **Dropbox** codifica, tanto su servidor como el software de escritorio, principalmente en Python.
- Una **Raspberry Pi**, lleva en un SO (eg. Raspbian) instalado Python, como lenguaje para aprender a programar.
- El sistema de intercambio de archivos peer-to-peer, **BitTorrent**, comenzó siendo un programa Python.
- **Industrial Light** & **Magic, Pixar** y otros usan Python en la producción de películas animadas.
- **ESRI** (*Environmental Systems Research Institute*) utiliza Python como herramienta de personalización para el usuario final de sus productos de cartografía SIG.
- El framework del desarrollador web **Google App Engine**, utiliza Python como lenguaje de aplicación.
- La **NSA** utiliza Python para la criptografía y el análisis de inteligencia.
- **Intel, Cisco, Seagate,** e **IBM** utilizan Python para pruebas de hardware.
- **JPMorgan, UBS, Getco y Citadel** aplican Python al pronóstico del mercado financiero.
- **NASA**, Los Álamos, Fermilab, JPL y otros, utilizan Python para la programación científica.

2.3. LIBRERÍAS

2.3.1. PARA EL PROCESAMIENTO DE IMAGEN

2.3.1.1. OpenCV

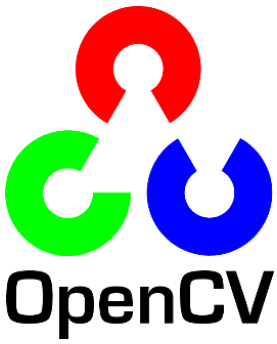


Figura 7: Logotipo actual de OpenCV[16]

Es una librería libre de visión artificial. Fue originalmente desarrollada por Intel. La primera versión alfa se lanza en enero de 1999[17]. Después fue financiado por Willow Garage, y en la actualidad, por Itseez[18].

Su publicación se da bajo licencia BSD (Berkley Software Distribution), lo que hace que sea fácil para las empresas utilizar y modificar el código.

[16] http://opencv.org/
[17] https://es.wikipedia.org/wiki/OpenCV
[18] https://en.wikipedia.org/wiki/OpenCV

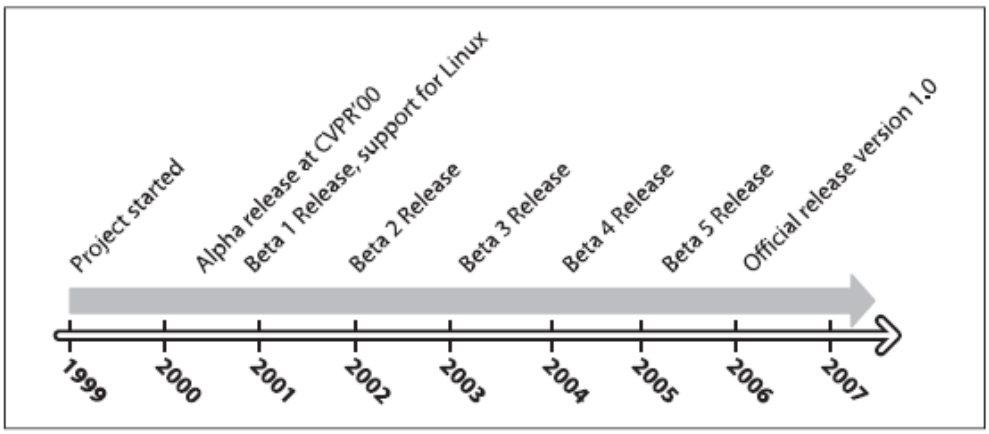


Figura 8: Línea temporal de la creación de OpenCV. [Bradski08]

La librería contiene más de 2500 algoritmos optimizados de visión por ordenador y de aprendizaje máquinas. En su web oficial[19], se describen algunas de sus aplicaciones:

- Detectar y reconocer rostros
- Identificar objetos
- Clasificar acciones humanas en vídeos
- Realizar un seguimiento de movimientos de cámara
- Rastrear objetos en movimiento
- Extraer modelos 3D de objetos
- Producir nubes de puntos 3D desde cámaras estéreo.
- Unir imágenes para mejorar la resolución de una imagen de una escena.
- Buscar imágenes similares en una base de datos.
- Quitar ojos rojos de las fotos tomadas con flash.
- Seguir los movimientos de los ojos.
- Reconocer el paisaje y establecer marcadores para superponerlo con realidad aumentada

Muchas empresas utilizan esta librería, entre las que se encuentran: Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota. Y muchas startups, por ejemplo: Applied Minds, VideoSurf y Zeitera.

Algunos de los usos empleados por las empresas son:

- Unión en imágenes en StreetView.
- Detección de intrusos mediante videovigilancia en Israel.
- Supervisión de equipos de minas en China.
- Ayuda a robots a navegar y recoger objetos en Willow Garage.
- Detección de accidente en piscinas.
- Inspección de etiquetas de los productos de las fábricas.
- Detección de rostros en Japón.

Es multiplataforma: tiene versiones para GNU/Linux, Mac OS X y Windows. Tiene interfaces para C++, Python, Java y MATLAB.

[19] http://opencv.org/about.html

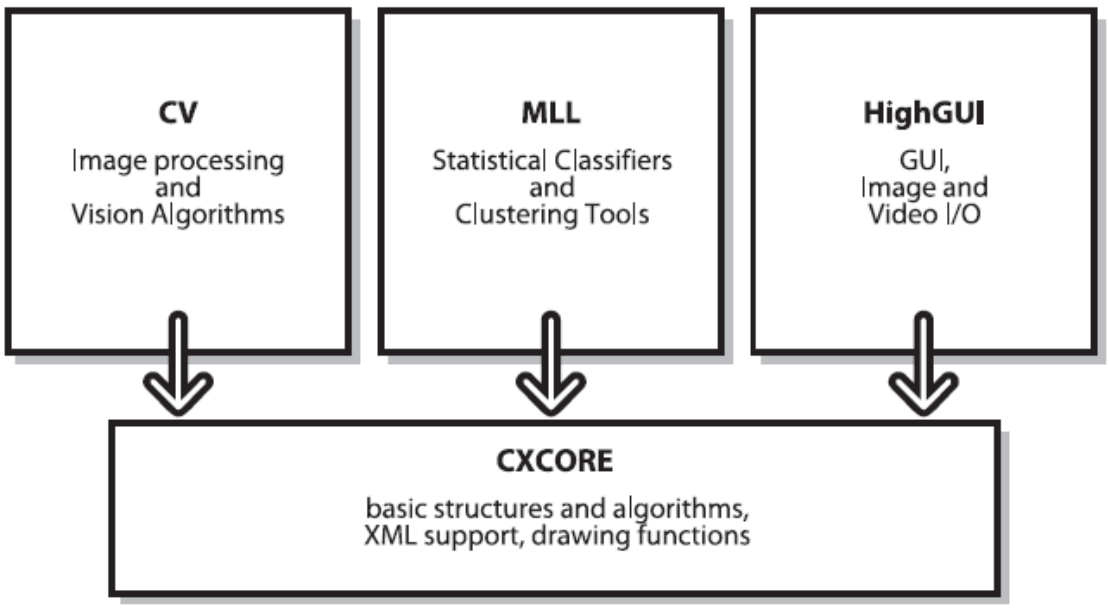


Figura 9: Estructura básica de OpenCV. [Bradski08]

2.3.2. LIBRERÍAS AUXILIARES

2.3.2.1. NumPy

NumPy[20] es un paquete usado en computación científica con Python.

Su versión anterior, Numeric, fue creada por Jim Hugunin en 1995, con la colaboración de otros desarrolladores. En 2005, Travis Oliphant creó Numpy incorporando características de la competencia Numarray en Numeric[21].

Contiene algunas funcionalidades útiles como los *array* de objetos, con los que representar vectores, matrices, imágenes y más; y funciones de algebra lineal.

Permite realizar operaciones importantes como la multiplicación de matrices, transposición, resolver sistemas de ecuaciones, multiplicación de vectores y normalización. Las cuales necesitamos para alinear imágenes, deformar imágenes, variaciones de modelado, clasificar imágenes, agrupar imágenes [Solem12] o Transformadas de Fourier.

NumPy también puede ser utilizado como un eficiente contenedor multidimensional de datos genéricos. Esto permite a NumPy integrarse sin problemas y rápidamente con una amplia variedad de bases de datos.

Numpy tiene licencia BSD, por lo que se permite la utilización con pocas restricciones.

2.3.2.2. Matplotlib

Matplotlib es una librería para la generación de gráficos, útil para trabajar con gráficas de matemáticas, o para dibujar puntos, líneas o curvas en imágenes.

[20] http://www.numpy.org/
[21] https://en.wikipedia.org/wiki/NumPy

Se creó en 2003. Fue escrito originalmente por John D.Hunter con la contribución de muchos desarrolladores. Los co-desarrolladores actuales son Michael Droettboom y Thomas A. Caswell[22].

Es una aplicación orientada a objetos. El módulo PyLab de Matplotlib es el conjunto de funciones que permitirán al usuario crear las gráficas. Proporciona una interfaz tipo MATLAB, especialmente cuando se combina con IPython[23].

2.4. ALGORITMOS RELACIONADOS

2.4.1. FEATURES (CARACTERÍSTICAS O RASGOS)

En OpenCV, las ***features*** son características que nos ayudan a identificar de mejor forma una imagen. Para identificar una porción de una imagen de su imagen total, se podrán atender a diferentes características, generales como puede ser el color, o a la continuidad.
Para entender este concepto, se utiliza la siguiente imagen en la documentación de OpenCV[24]:

Figura 10: Ejemplo de un edificio, y seis piezas pertenecientes a la imagen, para entender qué son las características.

Se expone una imagen en grande, con distintas partes de esa misma imagen contenidas en seis recuadros. Los cuadros A y B contienen imágenes planas, o un mosaico. Es difícil identificar con exactitud su lugar correspondiente en la imagen en grande. C y D son piezas más fáciles de encajar, ya que contienen más información. Aun así, es complicado determinar exactamente donde se sitúan, debido a su repetición (aparente) en la imagen grande. Sin embargo, E y F sí

[22] Créditos en la página Oficial de MatplotLib. http://matplotlib.org/users/credits.html
[23] Página official de MatplotLib. https://matplotlib.org/
[24] http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_meaning/py_features_meaning.html#features-meaning

contienen suficiente información. Estos recuadros contienen esquinas *(corners*), de las partes del edificio, que no se repiten a lo largo de la imagen.

Estos *features,* son patrones específicos que son únicas y pueden compararse fácilmente. Aunque en este ejemplo coincide con esquinas, en realidad pueden ser imágenes curvas.

Para identificar estos ***corners,*** en una imagen, tenemos que atender al hecho de que estas regiones tienen variación de intensidad máxima respecto a las regiones que la rodean. Esto se denomina Feature Detection (Detección de Características).

Para encontrar estas características en otras imágenes, se tiene que tener en cuenta la región que rodea estas esquinas. Este proceso se llama Feature Description (Descripción de características).

2.4.2. HARRIS CORNER DETECTION

Para detectar estas esquinas, Chris Harris y Mike Stephens escribieron un artículo llamado *A Combined Corner And Edge Detector* en 1988[25].[Harris88]

El procedimiento es encontrar la diferencia de la intensidad de un punto en (u, v) en todas las direcciones, a lo largo de un desplazamiento[26].

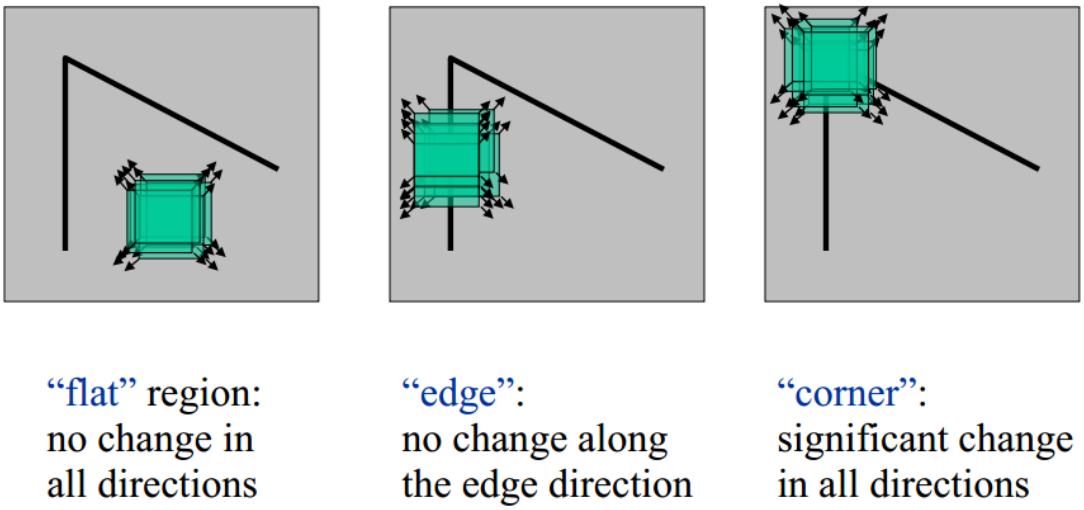


Figura 11: Ejemplo de las situaciones del algoritmo de Deteccion de Esquinas Harris.[27]

Utilizando la siguiente fórmula:

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \; [\underbrace{I(x+u,y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2$$

La cual se puede dividir en dos términos. El primer término, *window function,* corresponde a una ventana rectangular o gaussiana que da pesos a los píxeles. Para maximizar la función se deberá maximizar el segundo término. Para ello se utiliza la Expansión de Taylor, y tras algunos pasos

[26] Documentación sobre el Algoritmo de Detección de Esquinas Harris. http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html#harris-corners
[27] https://dsp.stackexchange.com/questions/14338/corner-detection-using-chris-harris-mike-stephens

matemáticos se llega a una matriz M que contiene las derivadas de la imagen en las direcciones x e y.

Para determinar si una imagen puede contener una esquina, se utilizará la siguiente ecuación.

$$R = \det(M) - k(\mathrm{trace}(M))^2$$

La cual utiliza los autovalores de la matriz M.

- Una imagen será plana si R es pequeño.
- Corresponderá con un borde si R < 0
- Se tratará de una esquina si R es grande.

Estas operaciones se pueden realizar con una función de la librería de OpenCV llamada cv2.cornerHarris(), cuya estructura en Python es:

```
cv2.cornerHarris(src, blockSize, ksize, k[, dst[,
borderType]]) → dst
```

Siendo:

- src: imagen de entrada en escala de grises y tipo float32
- blockSize: el tamaño del vecino (imagen próxima) considerado para la detección de la esquina
- ksize: el tamaño de apertura del operador de Sobel.
- k: parámetro de la fórmula anterior.

La función Sobel se utiliza para calcular la matriz M:

```
cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[,
borderType]]]]]) → dst
```

Para detector las esquinas con mayor precisión se puede utilizar la función cv2.cornerSubPix():

```
Cv2.FindCornerSubPix(image, corners, win, zero_zone, criteria)
→ corners
```

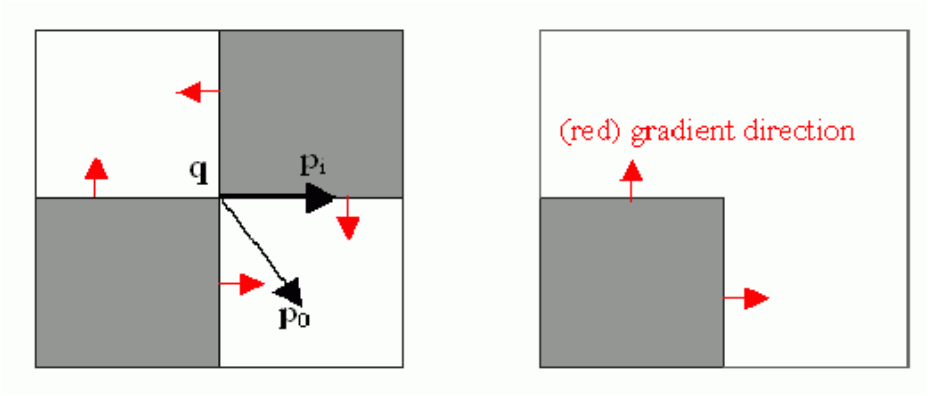


Figura 12: Explicación gráfica de la detección subpixel.[28]

[28] OpenCV. Feature Detection.
http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html

Para ello se hace el producto vectorial de un punto aproximado de la esquina, y varios puntos a su alrededor. Para po (p en cualquier sitio) y pi (p en un borde) se cumple la siguiente ecuación:

$$< \nabla I(p), q - p > = 0$$

Se forma un sistema de ecuaciones iguales a cero. En la resolución se obtendrá una esquina con mayor precisión.[29]

2.4.3. DETECTOR DE ESQUINAS SHI-TOMASI

En 1994, Jianbo Shi y Carlo Tomasi publican su artículo *Good Features To Track* [Shi94]. En él, modifican la ecuación de puntuación del Harris Corner Detector[30], donde:

$$R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)^2$$

Se redefine como:

$$R = \min(\lambda_1, \lambda_2)$$

Se establece un umbral mínimo, y si R es mayor, la región será considerada como una esquina.

La librería OpenCV[31] contiene la función para encontrar las N esquinas más fuertes en una imagen:

```
cv2.goodFeaturesToTrack(image, maxCorners, qualityLevel,
minDistance[, corners[, mask[, blockSize[, useHarrisDetector[,
k]]]]]) → corners
```

Donde:

- **image**: deberá ser una imagen en escala de grises

- **maxCorners**: el número máximo de esquinas que se desean encontrar

- **qualityLevel**: especifica la calidad mínima de la esquina para ser aceptado. Este valor está comprendido entre 0 y 1.

- **minDistance:** la mínima distancia euclidiana que tiene que haber entre las esquinas que se devuelven.

2.4.4. INTRODUCCIÓN A SIFT (TRANSFORMACIÓN DE CARACTERÍSTICAS INVARIANTE A ESCALA)

Los métodos de Harris y Shi-Tomasi encontrarán las mismas esquinas en una imagen, y en esa misma imagen rotada. Pero sí tendrán problemas para detectar las mismas esquinas con una imagen que varíe su tamaño.

La documentación ofrece el ejemplo siguiente:

[29] AI Shack. http://www.aishack.in/tutorials/subpixel-corners-increasing-accuracy/
[30] Documentación sobre el algoritmo Shi-Tomasi. http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html#shi-tomasi
[31] http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html

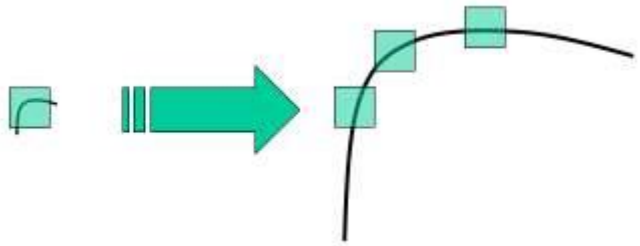


Figura 13: detección de una esquina, y no-detección cuando se amplía una imagen.[32]

En la izquierda, se representa una esquina que ha sido detectada, con unos parámetros establecidos. A la derecha, se muestra la imagen ampliada, y el efecto que tendría el algoritmo con los mismos parámetros y ventana. Como se aprecia, se encuentra continuidad en las secciones, y por tanto no se identifica como una esquina.

David G. Lowe crea en 2004 un algoritmo para resolver este problema. Lo publica en su artículo [Lowe04]. En él, se detallan los siguientes pasos.

El **primero**: detección de extremos. Se deberá utilizar un tamaño de ventana adecuado. Para ello se hace una aproximación del Laplaciano de Gauss, (ya que éste es algo costoso). En su lugar, se realiza la resta de la imagen borrosa y la imagen borrosa de sus adyacentes, y el proceso se repite para diferentes octavas de la imagen:

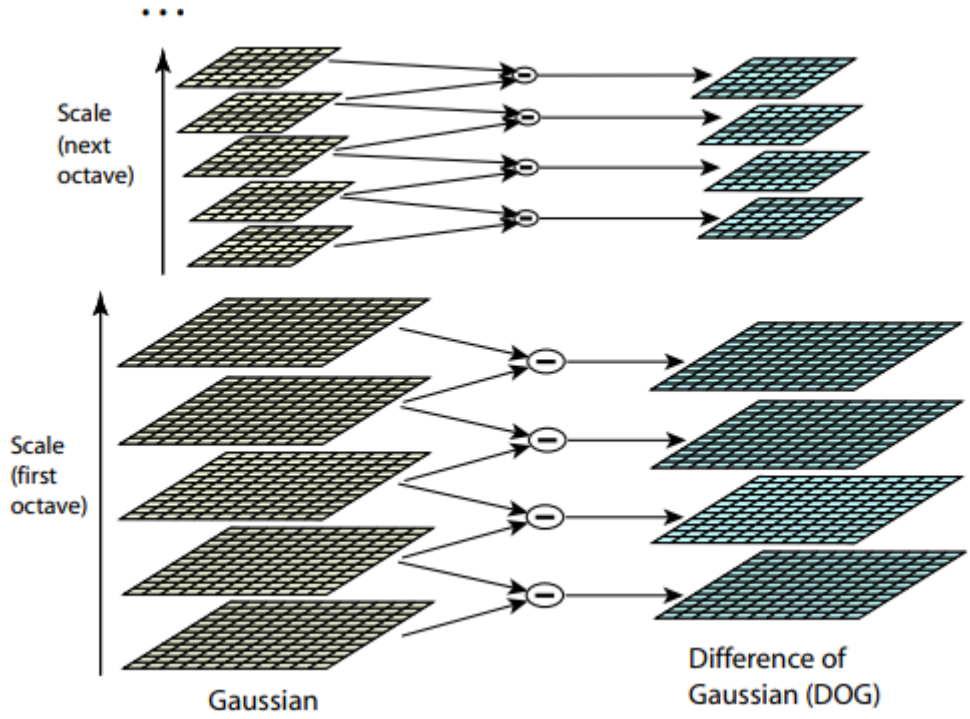


Figura 14: Funcionamiento de SIFT: resta de gaussianas para diferentes escalas.

Después se buscan los extremos locales.

[32] Documentación sobre el Algoritmo SIFT. http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html#sift-intro

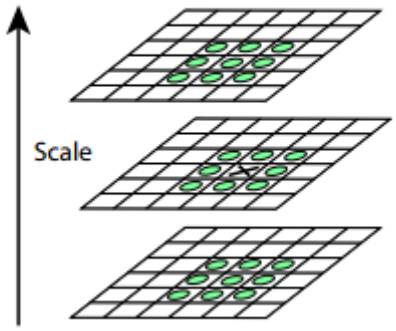


Figura 15: Búsquedas de maximos y mínimos. Representación del pixel candidato (X) y sus «vecinos» (O).

Los máximos y los mínimos de las diferencias gaussianas, realizas anteriormente, se detectan comparando un píxel (marcado con una X) con sus 26 vecinos (representados con círculos) en regiones, en las escalas actual y adyacentes.

En **segundo** lugar, se localizan los Puntos Clave. Para ello, se realiza una expansión en serie de Taylor del espacio de escala para obtener la localización más exacta del máximo o mínimo. Se establece un umbral de intensidad, y si la intensidad del extremo es menor, es decir, tiene bajo contraste, se rechaza.

Para los bordes, se recurre a una matriz Hessiana 2x2, y se comparan los autovalores de la matriz. Si la relación supera cierto umbral edgeThreshold, el punto clave se descarta.

El **tercer** paso es la Asignación de Orientación. Se realiza para lograr una invarianza a la rotación de la imagen. Se toma punto vecino alrededor del punto clave, y se calcula la dirección en esa región. Se crea un histograma con 36 rectángulos que representan los 360 grados. Se toma cualquier pico que supere el 80% de la orientación y el pico máximo, y se crean los puntos clave con misma ubicación y escala, pero diferente dirección.

En **cuarto** lugar, se crea un Descriptor de Puntos Clave. Para ello, se seleccionan los 16x16 vecinos alrededor de un punto clave, que a su vez se dividen en sub-bloques 4x4. Para cada sub-bloque se crea un histograma de orientación de 8 rectángulos.

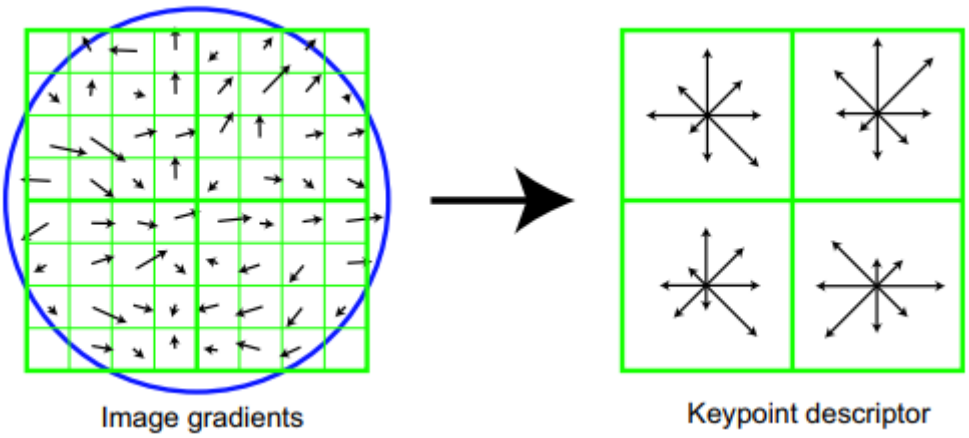


Figura 16: Explicación gráfica del descriptor.

El descriptor estará representado con un vector. En el mismo estudio, se realizan diferentes medidas teniendo en cuenta la sensibilidad a los cambios de rotación o iluminación.

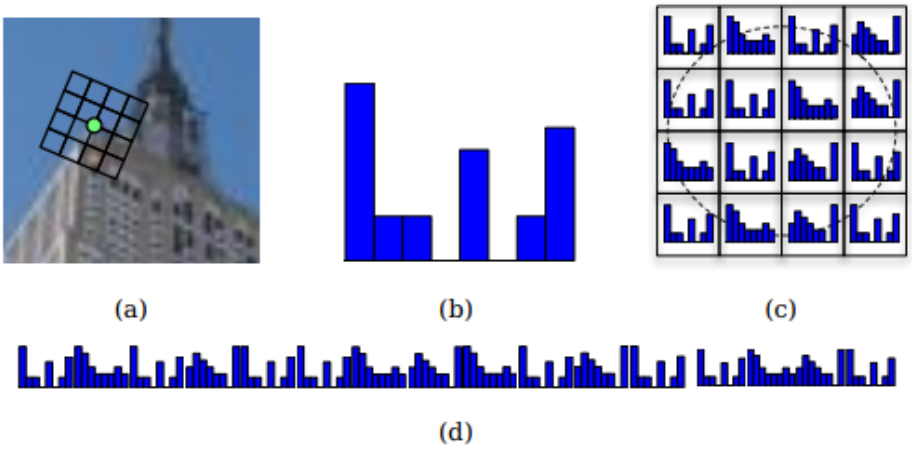


Figura 17: Ilustración del constructor del vector de características del descriptor SIFT. (a) un fotograma sobre un punto candidato, orientado de acuerdo la dirección de gradiente dominante. (b) un histograma de 8 rectángulos que indican la dirección del gradiente en un cuadro de la matriz (c) los histogramas de cada cuadro. (d) histogramas concatenados formando un vector de características. [Bradski08]

El mejor match (coincidencia) de candidatos para cada punto clave, se encuentra identificando a su vecino más cercano (*nearest neigbor*). Éste estará a la distancia mínima euclídea para el vector descriptor. [Lowe04].

Pero puede darse el caso de que, la segunda coincidencia más cercana se encuentre muy cerca de la primera. Para ello, se establece un ratio entre la distancia de la coincidencia más cercana, y la distancia de la segunda coincidencia más cercana. Si es mayor de que, por ejemplo, 0.8 (segundo punto muy cerca), se descarta.

Para utilizar SIFT en OpenCV, primero ha de crearse un objeto SIFT, de la siguiente forma[33]:

```
sift = cv2.xfeatures2d.SIFT_create()
```

Después, para encontrar los puntos clave, se aplica la función detect() a este objeto[34]:

```
sift.detect(image[, mask]) → keypoints
```

Donde:

- **image** será la imagen donde se quieren encontrar los puntos clave (tendrá que estar en escala de grises)
- **mask:** si se quiere buscar en una parte de la imagen. Si no, se escribirá None.

Si se quieren dibujar las ubicaciones de los puntos sobre la imagen, OpenCV dispone de la función cv2.drawKeyPoints.

Para calcular los descriptores se utiliza la función compute, que se aplicará al objeto sift, que devuelve los puntos clave y el descriptor. Se llama de la siguiente manera:

[33] Documentación de OpenCV. http://docs.opencv.org/trunk/da/df5/tutorial_py_sift_intro.html
[34] http://docs.opencv.org/2.4/modules/nonfree/doc/feature_detection.html

```
Kp, des = sift.compute(gray, kp)
```

Siendo gray la imagen en escala de grises, y kp una lista de puntos clave. des es un array numpy de la forma (número de puntos clave)*128.

También se puede buscar los puntos clave y los descriptores en la misma función de la forma:

```
Kp, des = sift.detectAndCompute(gray,None)
```

2.4.5. INTRODUCCIÓN A SURF (SPEEDED-UP ROBUST FEATURES)

Se conoce como una versión acelerada de SIFT, el cual se consideraba relativamente lento. Herbert Bay, Tinne Tuytelaars, y Luc Van Gool publican en 2006 el artículo **SURF**: Speeded Up Robust Features. [Bay06]. En este algoritmo se aproxima al Laplaciano de Gauss con Box Filter (en lugar de con la Diferencia de Gauss). Éste ofrece dos ventajas:

La convolución con Box Filter se puede calcular fácilmente con ayuda de imágenes integrales. Se puede hacer en paralelo para diferentes escalas.

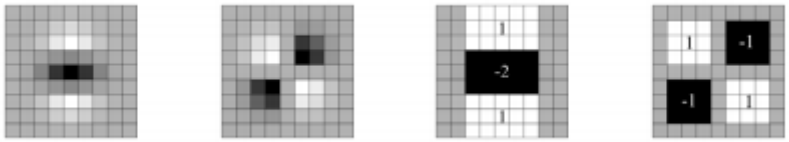


Figura 18: Box filter. Las zonas grises son iguales a cero. De izquierda a derecha: las dos primera imágenes se corresponden a las derivadas parciales de Gauss en segundo orden (discretizadas y recoradas) en la dirección y y xy. Las otras dos imágenes corresponden a las aproximaciones usando Box Filter [Bay06].

Para obtener un punto invariante a la rotación, orientación e iluminación se utilizan respuestas wavelets de Haar en dirección horizontal y vertical, en una región circular de radio 6*s, siendo s la escala del punto de interés.

La orientación principal se determina calculando la suma de todas las respuestas dentro de la ventana, de ángulo 60º. El vector más largo definirá la orientación del punto de interés.

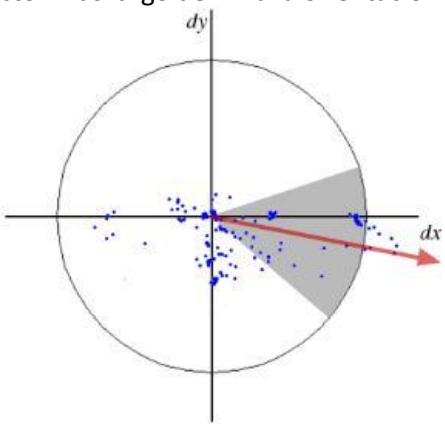


Figura 19: Explicación gráfica de la obtención de la orientación del descriptor de SURF.[35]

[35] Documentación de OpenCV sobre SURF.
http://docs.opencv.org/trunk/df/dd2/tutorial_py_surf_intro.html

Para el cálculo del descriptor, se construye una región cuadrada centrada en el punto clave y con tamaño 20sx20s. Ésta se subdivide en 4x4 subregiones, y en ellas se calculan las características simples, el Wavelet de Haar para las componentes x e y, las respuestas de onda horizontal y vertical. Se suavizan los resultados con un filtro Gaussiano, obteniendo dx y dy. Con esto, se puede obtener un vector de esta forma:

$$\mathbf{v} = (\sum d_x, \sum d_y, \sum |d_x|, \sum |d_y|).$$

Representado con un vector da lugar al descriptor de características SURF, con un total 64 dimensiones.

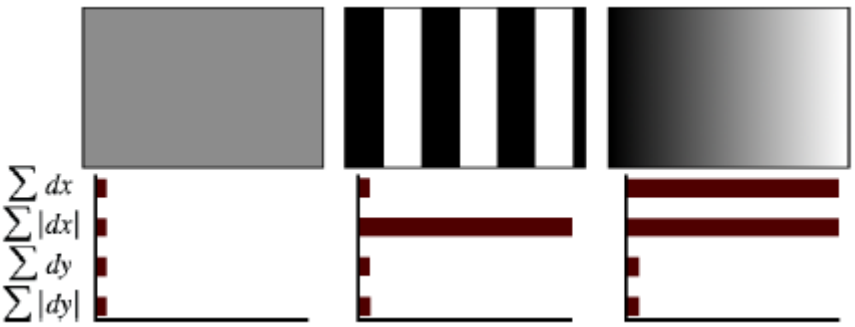


Figura 20: Representación de las componentes de la ecuación 5, con imágenes de ejemplo y sus histogramas. [Bay06]

El signo del Laplaciano distingue manchas brillantes sobre fondos oscuros. Cuando se produce una coincidencia se compara si las características tienen el mismo tipo de contraste:

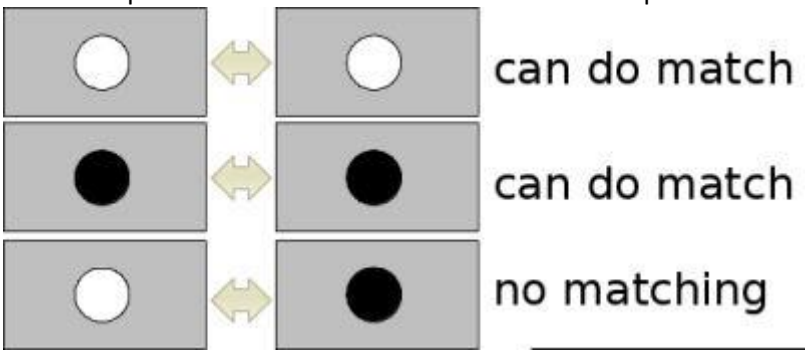


Figura 21: Tres diferentes tipos de contraste. Para los dos primeros sí hay coincidencia, y para el tercero no.

SURF mejora la velocidad respecto a SIFT, pero el rendimiento es similar, según se muestra en estudios como la figura siguiente:

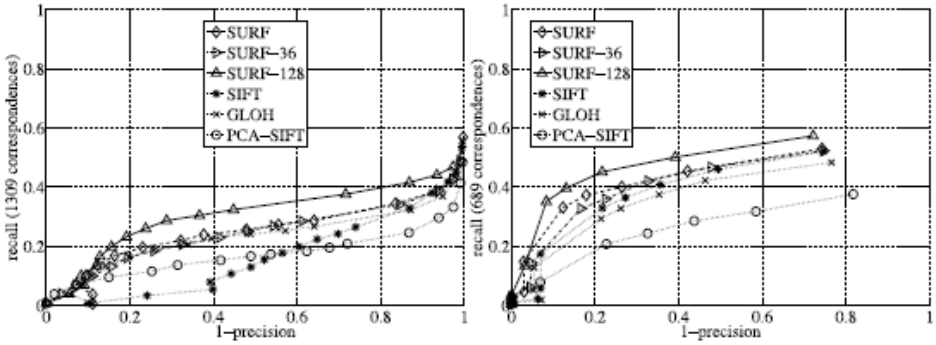


Figura 22: Comparativa del Recall en función de la precisión con SURF, SIFT, GLOH y PCA-SIFT [Bay06]

Para ejecutar el algoritmo SURF sobre una imagen, podemos contar con algunas funciones de la librería OpenCV.

Al igual que SIFT, primero se ha de crear un objeto[36]:

```
surf = cv2.xfeatures2d.SURF_create
```

Para hallar los puntos clave y el descriptor, se utiliza el mismo método que en SIFT:

```
kp, des = surf.detectAndCompute(img,None)
```

Para modificar el umbral, se utiliza la función:

```
setHessianThreshold ( double hessianThreshold )
```

Donde hessianThreshold es un número de tipo double.

Si la orientación no supone un problema, se puede usar el algoritmo U-SURF. Para activar el flag, se utiliza la función:

```
setUpright ( bool upright )
```

Donde upright deberá ser True.

Para cambiar el tamaño del descriptor, de 64 dim a 128 dim, se puede hacer utilizando la función:

```
setExtended(bool extended)
```

Donde extended deberá ser True.

2.4.6. ALGORITMO FAST PARA DETECCIÓN DE ESQUINAS

Edward Rosten y Tom Drummond publican, en 2006, el artículo *Machine learning for high-speed corner detection* [Rosten06], como respuesta a las demandas de un algoritmo más rápido. Los algoritmos SIFT y SURF no eran suficientemente rápidos para aplicaciones a tiempo real.

A continuación, se describen los pasos que realiza el algoritmo[37].

En primer lugar, se selecciona un pixel P, candidato a ser un punto clave, y se determina su intensidad Ip.

Se crea un círculo de 16 píxeles alrededor del píxel candidato.

[36] Documentación de OpenCV sobre el Algoritmo SURF.
http://docs.opencv.org/trunk/df/dd2/tutorial_py_surf_intro.html
[37] Documentación de OpenCV para FAST.
http://docs.opencv.org/trunk/df/d0c/tutorial_py_fast.html

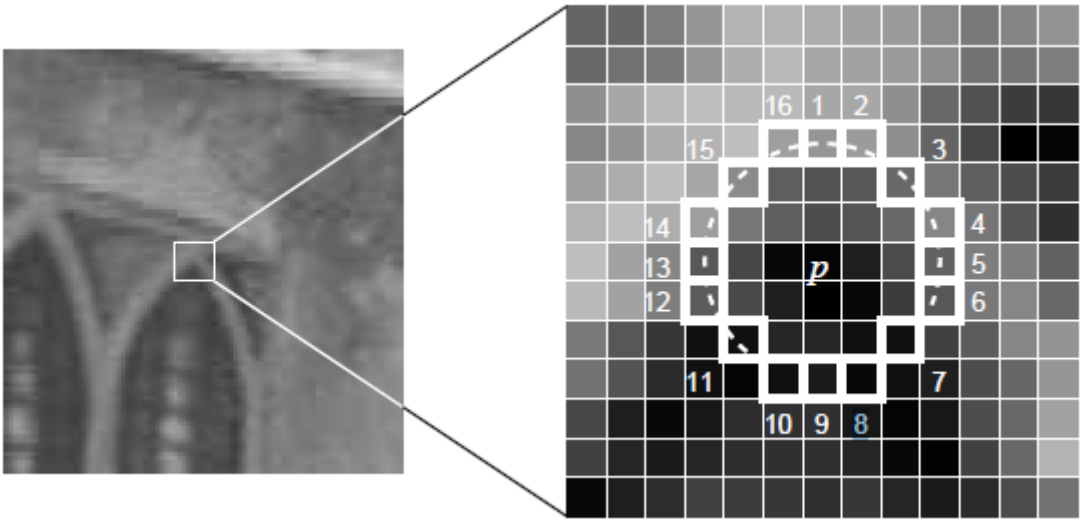


Figura 23: Ejemplo del algoritmo FAST en una imagen de una ventana. [Rosten06]

Se determina un umbral, t. Para clasificar el píxel como punto clave, se comprueba si existe un grupo de píxeles contiguos que forman el círculo que cumpla alguna de las siguientes condiciones: si son más brillantes que Ip + t, o más oscuros que Ip – t. Si existe un número n mayor de 12 que cumplan la condición, se considera como esquina.

Conviene prestar atención a los cuatro puntos en los extremos de la componente vertical y horizontal. En la imagen, éstos son: 1 y 9; y 13 y 5. Para considerar el píxel p como una esquina, al menos 3 de estos puntos deberán ser más brillante que Ip + t, o más oscuros que Ip – t.
Este algoritmo presenta algunas desventajas a pesar de tener alto rendimiento:

1. La prueba de velocidad no se generaliza bien para n < 12.
2. La elección y ordenación de los píxeles contiene suposiciones implícitas sobre la distribución de la característica.
3. Se descartan los resultados de las primeras cuatro pruebas de alta velocidad.
4. Se detectan múltiples características adyacentes entre sí.

El cuarto punto se trata usando la supresión no máxima. El resto, con aprendizaje automático. Para aplicar Aprendizaje Automático a este Detector de Esquinas, se hace en dos etapas, divididas en varios pasos.

Primero, se selecciona un **conjunto de imágenes para el entrenamiento** (preferiblemente del ámbito de la aplicación de destino).

Después **se ejecuta el algoritmo FAST** en todas las imágenes para encontrar los puntos clave, y para cada uno, almacena los 16 píxeles de su alrededor como vector.

Cada uno de estos píxeles se puede clasificar en uno de los siguientes estados:

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \le I_p - t \quad \text{(darker)} \\ s, & I_p - t < I_{p\to x} < I_p + t \quad \text{(similar)} \\ b, & I_p + t \le I_{p\to x} \quad \text{(brighter)} \end{cases}$$

De este modo, el vector se puede subdividir en conjuntos Pd, Ps, Pb.

A continuación, **se define una variable booleana**, Kp, que tendrá el valor True si p es una esquina, y falso en caso contrario.

En la **segunda etapa**, se utiliza el algoritmo ID3, un clasificador de árboles de decisión. Éste comienza seleccionando la x que da mayor cantidad de información sobre si el pixel p candidato es una esquina. Esta información está medida por la entropía de Kp, y se define como:

$$\begin{aligned} H(P) &= (c+\bar{c})\log_2(c+\bar{c}) - c\log_2 c - \bar{c}\log_2 \bar{c} \\ \text{where} \quad c &= \left|\{p|K_p \text{ is true}\}\right| \quad \text{(number of corners)} \\ \text{and} \quad \bar{c} &= \left|\{p|K_p \text{ is false}\}\right| \quad \text{(number of non corners)} \end{aligned}$$

Esto se aplica recursivamente a todos los subconjuntos. El proceso termina cuando la entropía de un subconjunto es cero.

El árbol de decisión creado se utiliza para la detección rápida de otras imágenes.

Para la solución del cuarto inconveniente de los cuatro expuestos, se utiliza la Supresión de los No-máximos. Se calcula una función de puntuación, V, para todos los puntos de clave detectados. Siendo V:

$$V = \max\left(\sum_{x\in S_{\text{bright}}} |I_{p\to x} - I_p| - t \;,\; \sum_{x\in S_{\text{dark}}} |I_p - I_{p\to x}| - t\right)$$

Se consideran dos puntos clave adyacentes y se calculan sus valores V. Se descarta el que tenga menor valor de V.

Para ejecutar este algoritmo en OpenCV, se crea el objeto de esta manera:

```
fast = cv2.FastFeatureDetector_create()
```

2.4.7. BRIEF (BINARY ROBUST INDEPENDENT ELEMENTARY FEATURES)

Michael Calonder, Vincent Lepetit, Christoph Strecha, y Pascal Fua, publican un artículo, *BRIEF: Binary Robust Independent Elementary Features* [Calonder10], como solución a los problemas de memoria y de tiempo que planteaban SIFT y SURF[38]. BRIEF proporciona un atajo para encontrar cadenas binarias sin tener que encontrar los descriptores. Es un descriptor de características (*features*), no un método para encontrarlas.
Se elimina tanto el clasificador como los árboles, y se crea un vector de bits nd de las respuestas del test, calculadas después de haber suavizado una porción de la imagen. Después, se compara la intensidad y se clasifica según la siguiente expresión:

$$\tau(\mathbf{p};\mathbf{x},\mathbf{y}) := \begin{cases} 1 & \text{if } \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0 & \text{otherwise} \end{cases},$$

Siendo p y q los pares de la localización.

[38] Documentación de OpenCV para el algoritmo BRIEF.
http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_brief/py_brief.html#brief

Una vez generado el vector nd, que por efecto tiene dimensión 256, aunque puede ser también 128 ó 512, se utiliza la distancia de Hamming para emparejar los descriptores.

Para utilizar este algoritmo en OpenCV, se crea el objeto con la siguiente función:

```
brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()
```

Y para usar el detector CenSurE citado en el artículo, se crea el objeto de la siguiente manera:

```
star = cv2.xfeatures2d.StarDetector_create()
```

Para conocer el tamaño del vector nd en bytes, se utiliza la siguiente función:

```
brief.getDescriptorSize()
```

2.4.8. ORB (FAST ORIENTADO Y BRIEF ROTADO)

En 2011, Ethan Rublee, Vincent Rabaud, Kurt Konolige y Gary Bradski publican *ORB: an efficient alternative to SIFT or SURF* [Ethan11]. Es una alternativa a SIFT y SURF en cuanto coste de computación y rendimiento.

Utiliza FAST para obtener los puntos clave, y el descriptor BRIEF con algunas modificaciones. Para encontrar los N mejores puntos, aplica el detector Harris. También se emplean pirámides para encontrar características multi-escala[39].

Para la orientación, se calcula el centroide de intensidad ponderada con el punto de esquina situado en el centro. La dirección del vector entre la esquina y el centroide indicará la orientación.

Para los descriptores, utiliza BRIEF. Originalmente, se sugiere el cálculo de un descriptor BRIEF para el conjunto de rotaciones y deformaciones de perspectiva de cada porción de la imagen. Pero esta solución es cara. Un método más eficiente será dirigir BRIEF de acuerdo a la orientación de puntos clave. Para cualquier conjunto de características de n pruebas binarias en la ubicación (xi, yi), se define la matriz S = 2 * n de la siguiente manera:

$$\mathbf{S} = \begin{pmatrix} \mathbf{x}_1, \ldots, \mathbf{x}_n \\ \mathbf{y}_1, \ldots, \mathbf{y}_n \end{pmatrix}$$

Usando la orientación θ, y su correspondiente matriz de orientación R_θ se obtiene la versión girada de S:

$$\mathbf{S}_\theta = \mathbf{R}_\theta \mathbf{S},$$

Después se discretiza el ángulo usando incrementos de 2π/30 (12 grados) y se construye una tabla de búsqueda de patrones BRIEF calculados antes. Mientras que en la orientación θ del punto clave sea consistente ente las vistas, el conjunto S_θ se utilizará para calcular el descriptor.

[39] Documentación de OpenCV para el algoritmo ORB.
http://docs.opencv.org/trunk/d1/d89/tutorial_py_orb.html

En cuanto a varianza y correlación, una propiedad de BRIEF es que cada bit de característica, tiene una gran varianza y su media es cercana a 0.5. En la siguiente figura se muestra la dispersión de medios para un patrón BRIEF de GAUSS de 256 bits en 100 000 puntos de muestra.

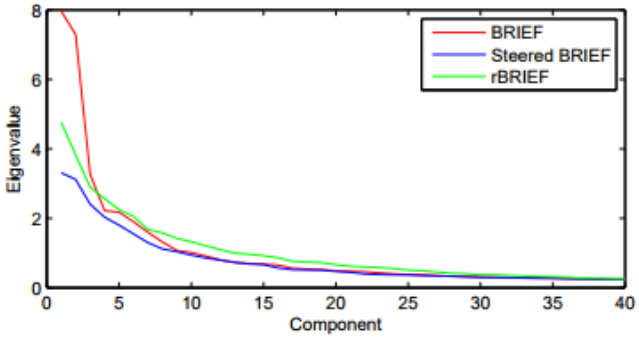


Figura 24: Distribucón de los valores propios en la descomposición de PCA para los algoritmos BRIEF, Steered BRIEF y rBRIEF [Rublee11]

La alta varianza hace que una característica sea más discriminativa, ya que responde de manera diferencial a las entradas.

Otra propiedad es que las pruebas serán no correlacionadas. ORB hará una búsqueda de un conjunto de pruebas no correlacionadas con media cerca de 0.5 y alta varianza. El resultado se denominará rBRIEF.

En la siguiente gráfica se comparan los rendimientos en función de los grados de rotación de la imagen.

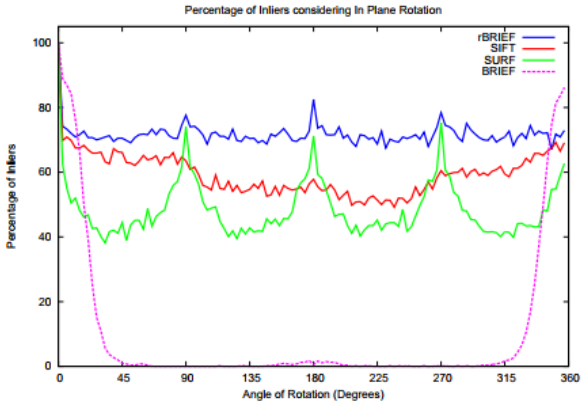


Figura 25: Rendimiento de las coindicencias de SIFT, SURF, BRIEF con FAST y ORB. Ruido gaussiano de 10. [Rublee11]

Para utilizar el algoritmo en OpenCV, se puede crear un objeto de la siguiente manera:

```
orb = cv2.ORB_create()
```

En la documentación[40], podemos encontrar los siguientes parámetros para ORB:

```
ORB_create([, nfeatures[, scaleFactor[, nlevels[,
edgeThreshold[, firstLevel[, WTA_K[, scoreType[, patchSize[,
fastThreshold]]]]]]]]) -> retval
```

[40] Documentación de OpenCV para el algoritmo ORB.
http://docs.opencv.org/trunk/d1/d89/tutorial_py_orb.html

Donde:

- **nFeatures:** es el número máximo de características a identificar. Por defecto: 500.
- **scoreType**: determina si se utiliza la puntuación de Harris o la de FAS para clasificar las características. Por defecto: puntuación de Harris.
- **WTA_K:** determina el número de puntos que genera cada elemento del descriptor BRIEF orientado. Por defecto: 2.

2.4.9. COINCIDENCIA DE CARACTERÍSTICAS (FEATURE MATCHING)

Un *matcher* (emparejador) de fuerza bruta toma el descriptor de una característica en la primera imagen, y lo relaciona con las características en la segunda imagen usando un cálculo de la distancia. Se selecciona aquella que se encuentre a menos distancia[41].

Para crear el objeto, se recurre a la siguiente función de la librería cv2 (OpenCV):

```
BFMatcher([, normType[, crossCheck]]) -> <BFMatcher object>
```

Donde:

- **`normType`:** determina la medida de la distancia a utilizar. Por defecto es: cv2.NORM_L2. Para SIFT y SURF se utiliza cv2.NORM_L1. Para ORB y BRIEF (cadenas binarias) se recurre a la distancia de Hamming, con el siguiente parámetro: cv2. NORM_HAMMING.

- **`crossCheck`:** es una variable booleana. Por defecto es: False. Si es True: devuelve aquellos *matches* (coincidencias) con valor (i,j). Esto es, el i-ésimo descriptor en el primer conjunto, que se relaciona con el j-ésimo descriptor del segundo conjunto, y vicerversa.

Después, se utiliza una función que devuelve la mejor coincidencia. Ésta tiene dos variantes:
BFMatcher.match(): devuelve la mejor coincidencia.
BFMatcher.knnMatch(): devuelve las k mejores coincidencias.

Si se quieren dibujar estos puntos sobre la imagen, se usan dos funciones:

```
cv2.drawKeypoints(image, keypoints[, outImage[, color[, flags]]]) →
outImage
```

Se utiliza para dibujar los puntos clave.

```
cv2.drawMatches(img1, keypoints1, img2, keypoints2, matches1to2[,
outImg[, matchColor[, singlePointColor[, matchesMask[, flags]]]]]) →
outImg
```

[41] Documentación de OpenCV sobre los Matchers.
http://docs.opencv.org/trunk/dc/dc3/tutorial_py_matcher.html

Para dibujar las coincidencias. Mostrará las dos imágenes en la misma pantalla, y una serie de líneas con un extremo en la coincidencia en la primera imagen, y el otro extremo en la coincidencia en la segunda imagen.

2.4.9.1 Emparejador de Fuerza Bruta con Descriptores ORB

Se intenta buscar una imagen queryImage, en una imagen trainImage.

Se crea el objeto BFMarcher:

```
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck = True)
```

Para crear el descriptor de las coincidencias, se utiliza la función match:

```
matches = bf.match(des1, des2)
```

Siendo des1 y des2 los descriptores de sendas imágenes. *matches* es una lista de objetos DMatch, y tiene los atributos:

- **DMatch.distance:** distancia entre los descriptores, y es mejor cuanto más bajo sea.
- **DMatch.trainIdx:** indica el descriptor en los descriptores de entrenamiento, o la imagen en la que se quiere buscar.
- **DMatch.queryIdx:** indica el descriptor en los descriptores de la imagen que se quiere buscar.
- **DMatch.imgIdx:** el índice de la imagen en la que se quiere buscar

2.4.9.2 Emparejador basado en FLANN

FLANN (Fast Library for Approximate Nearest Neighbors) [Muja09]. Contiene una colección de algoritmos optimizados para la búsqueda rápida de los «vecinos más cercanos» (*nearest neighbors*), en conjuntos grandes y muchas características.

En la implementación en Python, se deben crear dos diccionarios que especifican el algoritmo que se utiliza, y sus parámetros relacionados.

Un ejemplo de implementación del primer diccionario es el siguiente (por ejemplo, para SIFT y SURF):

```
Index_params = dict(algoritm = FLANN_INDEX_KDTREE, trees = 5)
```

Aunque existen más parámetros, descritos en la documentación, en `flann_algorithm_t`. Para ORB, se tienen en cuenta más parámetros. Un ejemplo:

```
Index_params = dict( algorithm = FLANN_INDEX_LSH, table_number
= 6, key_size = 12, multi_probe_level = 1)
```

El segundo diccionario especifica el número de veces que los árboles de índice deben recorrerse recursivamente. Un ejemplo:

```
search_params = dict(checks = 100)
```

Un mayor número de trees puede aumentar la precisión, en detrimento de la velocidad.

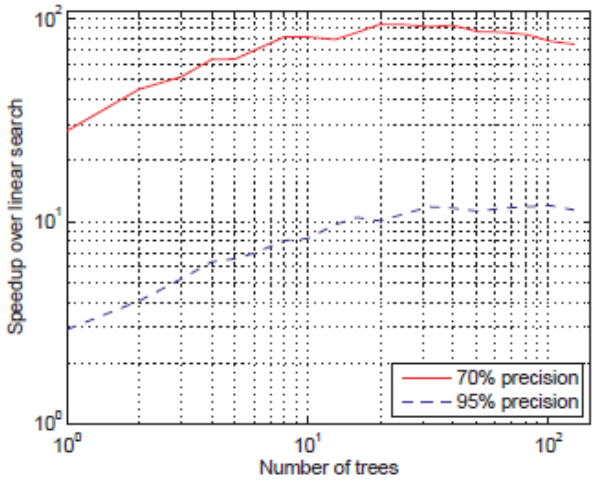


Figura 26: Comparativa de la velocidad en función del número árboles, para un 70% y un 95% de precisión. [Muja09]

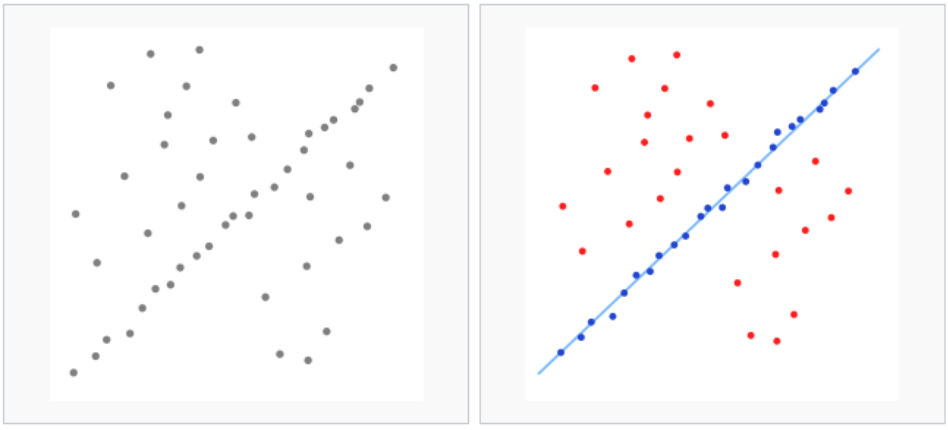
2.4.10. FEATURE MATCHING WITH HOMOGRAPHY

Se puede combinar la Coincidencia de Características con Homografía[42]. Para lo segundo, se utilizará la función findHomography. Esta función puede encontrar el objeto de una imagen (queryImage) en otra imagen (trainImage), a través de un conjunto de puntos.

Se ayuda de la función perspectiveTransform(), y necesita al menos cuatro puntos correctos para encontrarla, sin embargo, puede tener algunos errores. Estos se pueden solucionar mediante flags, que activan los atributos RANSAC o LEAST_MEDIAN. De esta forma, las buenas coincidencias serán llamadas inliers, y el resto, outliers.

```
cv2.findHomography(srcPoints, dstPoints[, method[, ransacReprojThreshold[, mask]]]) → retval, mask
```

El vector *mask* nos devuelve una cadena de 1s y 0s, siendo los 1s los inliers y los 0 los outliers. En funcionamiento de RANSAC se describe en la siguiente imagen:



Conjunto de datos con muchos valores atípicos.

Línea creada con RANSAC; los valores atípicos no tienen ninguna influencia en el resultado.

Figura 27: funcionamiento de RANSAC[43]

[42] Documentación de OpenCV sobre detección de objetos con homografía. http://docs.opencv.org/trunk/d1/de0/tutorial_py_feature_homography.html

[43] https://es.wikipedia.org/wiki/RANSAC

3. ENTORNO SOCIO-ECONÓMICO Y MARCO REGULADOR

3.1. MATERIALES EMPLEADOS EN EL PROYECTO

3.1.1. ELEMENTOS HARDWARE

El **ordenador portátil** empleado tiene las siguientes características:

- **Modelo**: Toshiba Satellite
- **Procesador**: Intel® Core™ i7-3610QM CPU @ 2.30GHz 2.30 GHz
- **Memoria instalada (RAM):** 8,00 GB
- **Tipo de sistema:** Sistema operativo de 64 bits, procesador x64

Servidor externo para pruebas y Backup

```
root@ubuntu-central:~# cat /proc/cpuinfo
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 45
model name      : Intel(R) Xeon(R) CPU E5-2630L 0 @ 2.00GHz
stepping        : 7
microcode       : 0x1
cpu MHz         : 1999.999
cache size      : 15360 KB
physical id     : 0
siblings        : 1
core id         : 0
cpu cores       : 1
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush mmx fxsr sse sse2 ss syscall nx pdpe1gb rdtscp lm constant_tsc
 arch_perfmon rep_good nopl eagerfpu pni pclmulqdq vmx ssse3 cx16 pcid sse4_1 ss
e4_2 x2apic popcnt tsc_deadline_timer aes xsave avx hypervisor lahf_lm vnmi ept
tsc_adjust xsaveopt arat
bugs            :
bogomips        : 3999.99
clflush size    : 64
cache_alignment : 64
address sizes   : 40 bits physical, 48 bits virtual
power management:
```

Figura 28: Características del Droplet.

Se ha contratado un **Droplet**[44] para almacenar el proyecto en el caso de pérdida o error, y para posibles ejecuciones de pruebas con vídeos de larga duración. El Droplet funciona como un ordenador, al cual se accede a través de Internet. Se pueden ejecutar instrucciones mediante programas como Putty. Para navegar por los directorios de forma más gráfica con WinSCP. Es útil para mover el programa y los archivos del ordenador personal al Droplet, y para extraer los ficheros de salida generados, del Droplet al ordenador personal.

[44] Digital Ocean: https://www.digitalocean.com/

3.1.2. ELEMENTOS SOFTWARE

- **Sistemas Operativos**: Windows 10 (en el ordenador personal), Linux (en el servidor)
- Reproductor VLC
- Microsoft Visual Studio 2017
- **Editor de texto:** Microsoft Word, Notepad++
- **Editor de imagen**: Paint
- GNU Octave 4.2.1
- WinSCP, Putty

3.2. JUSTIFICACIÓN DEL ENTORNO DE DESARROLO

3.2.1 SISTEMA OPERATIVO

Para realización de este proyecto se ha utilizado **Windows 10**, debido al mejor conocimiento y compatibilidad con otros programas. Una desventaja de Windows, es que la instalación de librerías, entre las que se incluye OpenCV, requería más pasos que en Linux, donde con unas pocas líneas de comandos puede instalarse la librería sin problemas. En un principio, para instalar la parte *contrib* de OpenCV, se requería crear un programa binario con CMake y el compilador de Visual Studio. Esto ya no es necesario, ya que existen versiones de Python, que tienen un programa llamado pip (generalmente en la carpeta Scripts en el directorio donde se ha instalado) el cual facilita la instalación de los paquetes. Se llama a través de la consola de Windows mediante `pip install nombre_paquete`. En el caso de no encontrar el paquete, se puede descargar el archivo .whl de la página de Archivos Binarios No Oficial de Paquetes Python para Windows[45], e instalarlo mediante pip.

3.2.2. ENTORNO DE PROGRAMACIÓN

Para aprender a utilizar Python se utilizó en un primer momento Sublime Text. Más tarde, por tener un entorno más parecido a otros conocidos (Eclipse), se utilizó Liclipse, un software gratuito. Al igual que programas similares, se muestra una ventana en un lateral con el árbol del programa. Para realizar un proyecto en el que se iban a utilizar imágenes, y obtener archivos de texto por la salida, se escogió este programa.

Por último, la mayoría del proyecto se realizó con el programa de **Microsoft, Visual Studio 2017**. La instalación incluía ya varios lenguajes, muchas herramientas, la integración de librerías era fácil, posibilidad de creación de proyectos grandes, y aunque Python no requiere compilación, incluye la opción de depuración del código. Microsoft IntelliSense, la cual ayuda a autocompletar algunas variables y funciones también ayuda.

Finalmente, en la creación del programa se tuvo en cuenta la facilidad de uso para el usuario final. Por ello, se ejecuta directamente desde la consola. Este modo de ejecución permitía probar el algoritmo con diferentes carpetas de imágenes y diferentes archivos de vídeo de forma rápida. Algunos cambios se hacían con Notepad++ para no perder demasiado tiempo.

3.2.3. LENGUAJE DE PROGRAMACIÓN

El lenguaje de programación escogido es **Python 3.6**, debido a que esta última versión ya tiene buen soporte, y muchas de las funcionalidades que tenía Python 2.7, el que ya se ha dejado de actualizar. Además de la compatibilidad con OpenCV 3.

Las ventajas de usar Python se describen en la sección 2.2.1, pero las cuestiones personales han sido un mejor conocimiento de la Programación Orientada a Objetos, que sea un lenguaje cada

[45] http://www.lfd.uci.edu/~gohlke/pythonlibs

vez más utilizado en este entorno, su fácil manejo, la ausencia de necesidad de compilación, y aprender un lenguaje nuevo

3.3. PRESUPUESTO

3.3.1. PLANIFICACIÓN

El proyecto se divide en cuatro bloques principales:

1. **Análisis del problema y recopilación de información**. Se identificará qué problema hay que solventar, cuáles son los puntos a tratar con más profundidad, y qué recursos actuales podrían ayudar a su resolución. Una vez claro, se recopilará toda la información relativa a estos recursos (Python, Visión Artificial, Open CV, publicidad…)

2. **Preparación del entorno**. Se deberá escoger un equipo suficientemente potente para la ejecución de pruebas con vídeos. Después instalar todos los programas necesarios para la edición de imágenes, vídeos y texto; así como entornos de programación, lenguajes y librerías.

3. **Creación del programa**. Este bloque engloba: los cursos realizados para comprender Python y las herramientas de OpenCV, la creación del código para identificar una imagen en otra imagen, identificar una imagen en un vídeo, varias imágenes en un vídeo. También las funciones adicionales: generación de textos de salida, creación de procesos y pruebas finales

4. **Redacción de la memoria**. Se expondrá de forma ordenada y con imágenes aquellas partes de la información recopilada en el punto 1, y las pruebas resultantes del punto 4.

ID ACT.	DESCRIPCIÓN	INICIO	JORNADAS
A	Identificación del problema	25/10/2016	33
B	Determinación del lenguaje, librerías y entorno a utilizar	28/11/2016	64
C	Recopilación de información (libros, vídeos, webs …)	01/02/2017	95
D	Configuración de Hardware	05/05/2017	10
E	Instalación de software	15/05/2017	18
F	Recopilación de vídeos	03/06/2017	5
G	Curso básico de Python	08/06/2017	12
H	Prácticas de ejemplos con librerías OpenCV	20/06/2017	14
I	Curso de interfaz gráficas	04/07/2017	11
J	Curso de Machine Learning	15/07/2017	60
K	Creación del código para fotogramas individuales	15/07/2017	5
L	Creación del código para un vídeo	20/07/2017	10
M	Creación del código para varios vídeos y funciones extra	30/07/2017	10
N	Pruebas iniciales	10/08/2017	4
O	Creación de procesos en el código	14/08/2017	6
P	Pruebas finales	20/08/2017	2
Q	Escrito de la memoria sobre el funcionamiento del programa	22/08/2017	9
R	Escrito del resto de la memoria	31/08/2017	22

Tabla 1: Tabla de actividades de las que consta el proyecto.

A continuación, se muestra un diagrama de Gant con las actividades de la Tabla 1.

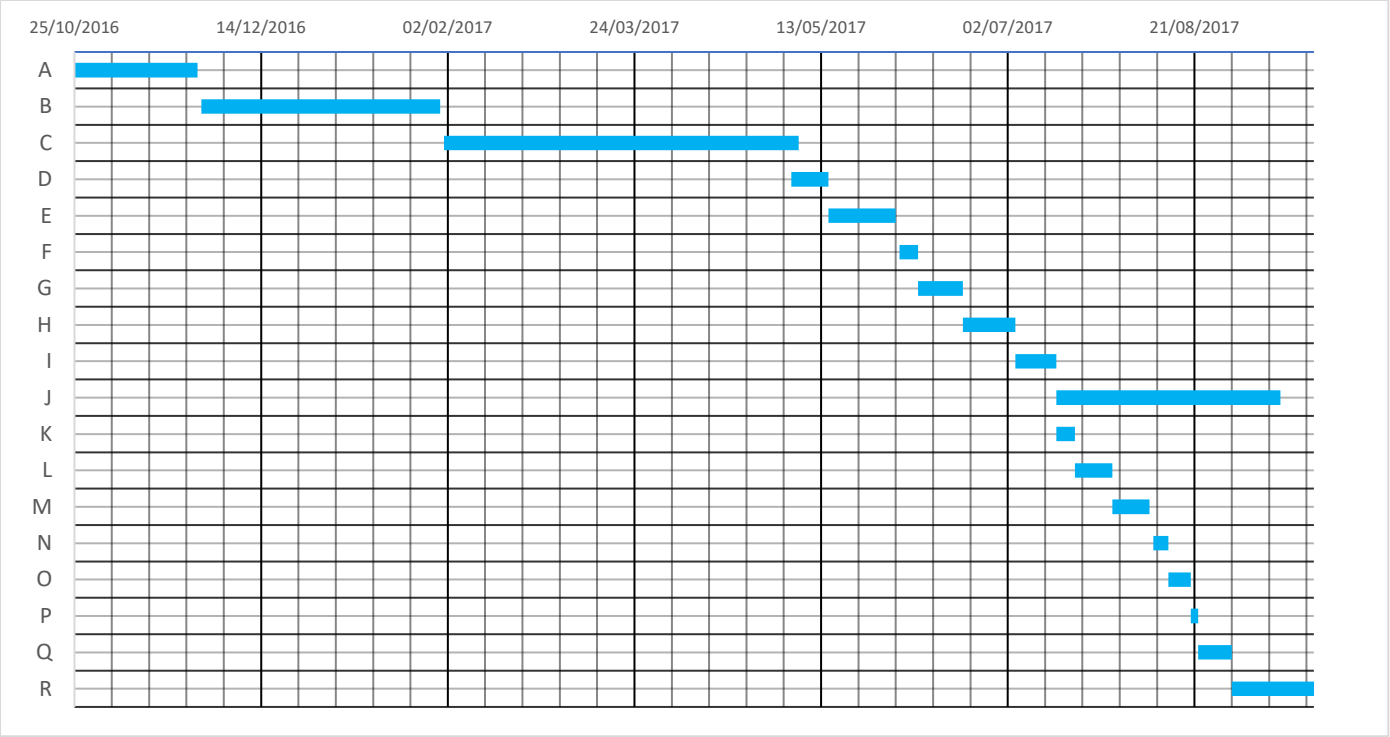


Figura 29: Diagrama de GANTT

3.3.2. COSTES DIRECTOS

7 horas/ jornada (40 h/semana máximo)

Sueldo Programador Junior: 1800 €/mes

	MESES	JORNADAS	SUELDO (€/MES)	TOTAL
PROGRAMADOR JUNIOR	11	55	800 €/mes	8800 €

Tabla 2: Sueldo de Programador Junior en España durante el proyecto

Vida útil del portátil: 5 años (60 meses).

$$Amortización\ anual = \frac{100}{8} = 12.5\ \%$$

En la siguiente tabla se incluye un resumen del gasto total:

RECURSO	COSTE UNITARIO (SIN IVA) (€)	DEDICACIÓN (MESES)	USO EN EL PROYECTO (%)	PERIODO DE DEPRECIACIÓN (MESES)	COSTE IMPUTABLE (€)
ORDENADOR PORTÁTIL	850	11	50	60	77.91
VISUAL STUDIO 2017	574	11	100	60	105.23
TOTAL					**183.14**

Tabla 3: Coste con amortiacion

$$C_{imputable} = \frac{T_{dedicacion}}{T_{depreciación}} * C_{unitario} * Uso$$

RECURSO	PRECIO UNITARIO (€)	CANTIDAD	TOTAL (€)
MICROSOFT OFFICE 365 PERSONAL	7	11	77
DROPLET	10	2	20
TOTAL			97

Tabla 4: Coste del Droplet y del software de pago

El precio del Droplet es:

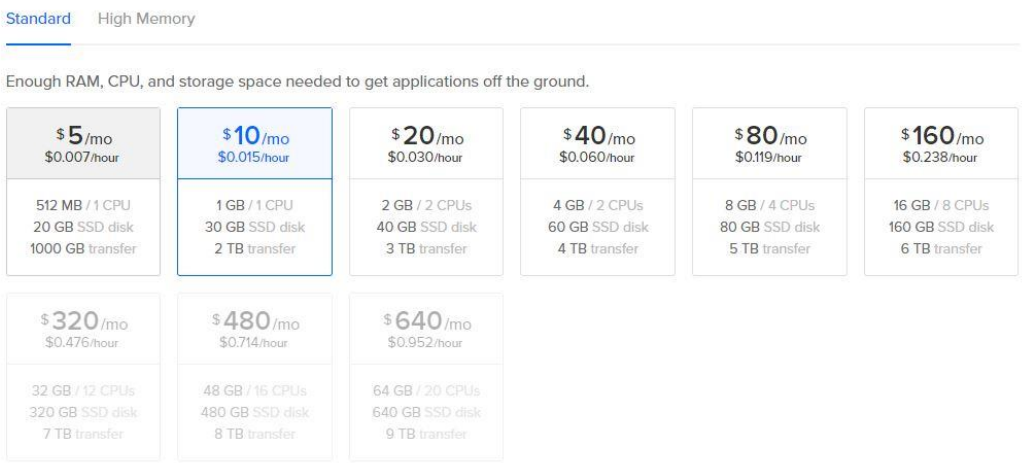


Figura 30: tarifas del Droplet de DigitalOcean

Como se señala en la sección 2.3 y 2.2.1, donde se habla de las librerías y el lenguaje utilizados, respectivamente, el software es libre.

Python se ha obtenido desde su página oficial. Las librerías han sido descargadas mediante el programa pip de Python (en la carpeta Scripts). Se han encontrado problemas de compatibilidad con algunas librerías. En estos casos se ha recurrido a las páginas citadas en el apartado de recursos. En especial a la página Unofficial Windows Binaries for Python Extension Packages[46].

Existen licencias de Microsoft Office y Microsoft Visual Studio para estudiantes de Universidad. Otros programas (VLC, ATube Catcher, WinSCP, PuTTy, Adobe Reader…) son gratuitos. Para la edición de imágenes se ha recurrido a herramientas online. E.g https://pixlr.com/editor/

Coste Directo Total: 9 080.14 €

3.3.3. COSTES INDIRECTOS

Los costes indirectos pueden la luz, Internet, muebles, calefacción, material de oficina, etc.

El Coste Indirecto suele ser 20 % del Coste directo, por tanto:

$$CI = CD * 0.2$$

Coste Indirecto Total: 1 816.02 €

[46] http://www.lfd.uci.edu/~gohlke/pythonlibs

3.3.4. RESUMEN DE LOS COSTES

COSTE	VALOR (€)
DIRECTO	9 080.14
INDIRECTO	1 816.02
TOTAL	**10 926.16**

Tabla 5: Coste total del proyecto

3.4. PLAN DE EXPLOTACIÓN Y BENEFICIO ECONÓMICO

El beneficio económico estará directamente relacionado con el ***Tracking de Publicidad,*** nacido en la era de la información, y como ayuda para manejar la gran cantidad de datos actual.

Ejecutar este algoritmo puede generar un informe breve y en poco tiempo de la frecuencia, duración, tiempo exacto de emisión, en el que se emite un anuncio. En un sistema automatizado. Existen muchos campos dónde un sistema automatizado ha supuesto una mayor velocidad y disponibilidad, con una probabilidad de aciertos similar al que tenía una persona. Se puede hacer una analogía con el ***player tracking***. Para contar los pases que realizan los jugadores, por ejemplo, en un partido de fútbol, las empresas contratan a jóvenes aficionados de estos deportes para que realicen el conteo[47]. Sin embargo, actualmente se desarrolla software automatizado para realizar estas mismas tareas[48]. Aquellos son sueldos que no habrá que pagar.

Comparar estos datos con las respuestas generadas puede indicar qué aspectos funcionan y cuáles no. Estas respuestas pueden ser medibles por muchas vías. Pueden ser las visitas generadas a la página web oficial, el aumento de las ventas de un producto, y herramientas recientes de análisis, como **Google Trends**. Un anuncio posiblemente funcione mejor si es emitido en determinada franja horario, si es incluido en el espacio publicitario de un programa en cuestión, o incluso si es mostrado entre anuncios que refuercen el mensaje. Factores como la repetición, la emisión en diferentes estaciones del año o los días de fiestas también pueden ser determinante al analizar la efectividad de un anuncio.

El coste de emisión de un anuncio en televisión es alto, por lo que es imperativo encontrar el máximo beneficio con el menor coste. Las propias cadenas lo saben, y valoran cada franja de televisión y cada evento emitido con su precio correspondiente. El conocer en qué puntos ha fallado la predicción de estas cadenas al fijar su precio, es la manera que tienen las empresas de generar beneficios.

3.5. MARCO REGULADOR

3.5.1. AVISO

El uso de los logos para realizar las pruebas en el programa, y representarlas en este documento, se ha realizado para fines docentes y de investigación, sin ningún ánimo de lucro. El registro de un logotipo da derecho a que su titular use en exclusiva el logo que esté identificado con unos productos y servicios determinados, siendo el titular el único legitimado para su explotación. Cualquier uso comercial del logo es totalmente denunciable por parte de su titular y corresponde un delito de propiedad industrial tipificado en el Código Penal.

[47] FiveThirtyEight. *The People Tracking Every Touch, Pass and Tackle in the World Cup* (2014). https://fivethirtyeight.com/features/the-people-tracking-every-touch-pass-and-tackle-in-the-world-cup/
[48] Xataka. Así se mide la distancia que recorre un futbolista durante un partido. (2014). https://www.xataka.com/otros/asi-se-mide-la-distancia-que-recorre-un-futbolista-durante-un-partido

SI alguna empresa o particular quisiese hacer uso de este programa con fines lucrativos, deberá adquirir el consentimiento de los propietarios de los logotipos.

3.5.2. PROPIEDAD INTELECTUAL (PI) EN LA PUBLICIDAD

En la página web de la Organización Mundial de la Propiedad intelectual[49], se describe como pueden proteger los anunciantes los derechos exclusivos sobre sus creaciones. Algunos de los tipos se citan a continuación.

3.5.2.1. Tipos de derechos de la propiedad intelectual en la publicidad

- «Los **contenidos creativos**: material escrito, fotografías, ilustraciones, gráficos, composición de un anuncio, música y vídeos». Protegidos por el derecho de autor.
- «Los **lemas** y **sonidos**».
- «**Signos** empleados en la publicidad: nombres comerciales, logotipos, nombres de producto y dominio».
- «Las **indicaciones geográficas** pueden estar protegidas por: la legislación contra la competencia desleal, denominaciones de origen» y otros.
- «**Símbolos gráficos**, visualizaciones de pantalla, interfaces gráficos de usuario y páginas web. Protegidas por la legislación en materia de diseño industrial».
- «**Programas informáticos** utilizados para crear anuncios digitales, y las **imágenes generadas por ordenador**. Protegidas por **derecho de autor** o por **patentes**».
- «**Técnicas publicitarias** o **métodos comerciales.** Protegidos por patentes o modelos de utilidad».
- «**Envase y embalaje.** Puede ser protegido como marca, diseño industrial o presentación comercial».
- «**Identidad de una persona.** Su nombre, fotografía, imagen, voz o firma. Protegida por derecho de publicidad o derecho a intimidad».

3.5.3. LEY 7/2010 DE 31 DE MARZO, GENERAL DE LA COMUNICACIÓN AUDIOVISUAL (LGCA) [50]

En el **Artículo 1** se describe el **Objeto de la Ley**: «Esta Ley regula la comunicación audiovisual de cobertura estatal y establece las normas básicas en materia audiovisual».

Defiende los derechos del público, los derechos de los prestadores de servicio de comunicación audiovisual. Contiene las **normas básicas** para la regulación y coordinación del Mercado de comunicación audiovisual.

En el caso de este proyecto, la mayoría de anuncios duran 20 ó 10 segundos. Desde la (LGCA), el tiempo del espacio publicitario puede ser de 20 minutos por hora, y se divide de la siguiente manera: 12 minutos por hora de reloj para publicidad; 5 minutos para las autopromociones, 3 minutos como máximo para las telepromociones y algunos segundos para patrocinio de programas[51]. Los anuncios durarán entre 5 y 30 segundos. Con algunas excepciones, como los anuncios de Freixenet[52].

[49]Página oficial de la OMPI (WIPO por sus siglas en inglés):
http://www.wipo.int/sme/es/documents/ip_advertising.htm
[50] https://www.boe.es/buscar/pdf/2010/BOE-A-2010-5292-consolidado.pdf

[51] (3/11/2014). ¿Cuál es el tiempo máximo de publicidad permitido? Qué Aprendemos Hoy.
http://queaprendemoshoy.com/cual-es-el-tiempo-maximo-de-publicidad-permitido/
[52] https://es.wikipedia.org/wiki/Freixenet

3.5.4. LICENCIA BSD

Como se ha explicado a lo largo del documento, las librerías de OpenCV y Numpy tienen licencia BSD. Es una licencia de software libre permisiva para los sistemas BSD (Berkeley Software Distribution). Permite el uso de código de fuente en software no libre. Existe una versión original y las llamadas licencias BSD modificadas[53].

[53] Wikipedia en español. Licencia BSD. https://es.wikipedia.org/wiki/Licencia_BSD

4. METODOLOGIA EMPLEADA

4.1. SELECCIÓN DE IMÁGENES

Para seleccionar las imágenes, se han tenido en cuenta varios factores:

- La imagen del logo de la empresa de publicidad, o de los canales, debe ser lo más parecida a la actual, o a la que se quiera detectar. Existen empresas que hacen pequeños cambios a sus logos, y otros que cambian radicalmente.

Figura 31: Evolución del logo de la cadena de televisión LaSexta.

En el ejemplo de laSexta. Cambió su logo en el 2007 debido a que las tres líneas, del anterior logo, provocaban una interferencia, el efecto de Moiré.

- Para una mejor detección de la imagen, es recomendable que contenga el **mismo fondo** en la imagen estática, que en la secuencia de vídeo. Por lo general, las marcas suelen tener unos colores característicos, ya no solo en el logo en sí, dentro de su forma, si no a su alrededor, y eso implica el mismo fondo. E.g. Coca Cola casi siempre aparecerá sobre un color fondo, ya sea en un anuncio de Televisión, en su web, merchandising o evento; o Telefónica, es raro no relacionar la empresa con el color azul.

Una vez escogidas las imágenes a priori, se deben seleccionar aquellas definitivas, con dos procesos:

- **Probar sus resultados**. Se ejecuta el programa con un conjunto de imágenes similares y se ejecuta el programa. De aquellas que den buenos resultados, se escoge la mejor. Si ninguna es apropiada, se repite el proceso con otro conjunto de nuevas imágenes.

- **Modificar la imagen**. Para el caso en el que ningún archivo de imagen dé buenos resultados, o simplemente se quiera mejorarlos, se puede tratar con OpenCV. Algunas de las opciones a considerar sería la ecualización del histograma, o la eliminación del ruido.

4.2. CORTES DE LOS VÍDEOS

4.2.1. SELECCIÓN DEL VÍDEO

El propósito del proyecto es detectar los anuncios en secuencias de anuncios. Para obtener estas secuencias se ha tenido en cuenta la calidad y la duración. Es importante que los fotogramas sean nítidos, que el vídeo sea fluido y sin errores debidos a la compresión. Respecto a la duración, se espera obtener una imagen completa del logo y al menos una transición entre anuncios (fotogramas en negro), para poder identificarlo.

4.2.2. VLC PLAYER

Se ha utilizado el reproductor multimedia VLC[54]. Es Software Libre y de código abierto multiplataforma. Fue creado como un proyecto en la École Centrale Paris en 1996. Según la web oficial, VLC es «un "framework" que reproduce la mayoría de archivos multimedia, así como DVD, Audio CD, VCD y diversos protocolos de transmisión».

Para reproducir el Streaming, se ha buscado un enlace con formato m3u8, y se ha utilizado VLC para reproducirlo mediante la herramienta *Abrir ubicación de red*.

Con el botón *Grabar,* en los controles avanzados, se han grabado 1 ó 2 horas de vídeo seguidas, y después se ha guardado el archivo de vídeo con formato .ts.

En el vídeo guardado, se puede manipular la barra de reproducción para identificar en qué momentos se emiten anuncios. Se ha vuelto a grabar, esta vez desde que un programa da paso a la publicidad, el tiempo de anuncios, y el inicio del programa. El vídeo resultante es que se utilizará finalmente para realizar las pruebas.

4.3 ELEMENTOS A BUSCAR

4.3.1. TABLA RESUMEN

RECURSO	NÚMERO DE ELEMENTOS
CANALES DE TELEVISIÓN	5
LOGOTIPOS	24
VÍDEOS	6

Tabla 6: Tabla resumen del número de los recursos utilizados.

[54] Página oficial de VLC Player: http://www.videolan.org/

4.3.2. VÍDEOS

#	CONTENIDO	NOMBRE FICHERO	FORMATO	DURACIÓN	ANCHO FOTOGRAMA	ALTO FOTOGRAMA	FPS
1	Anuncio de Booking	prueba-booking	TS	00:00:21	1280	720	25
2	Anuncio de Aldi	booking-aldi2	TS	00:00:45	1280	720	25
3	Anuncio de Juver	anuncio-juver	TS	00:00:09	1280	720	25
4	Anuncio Aldi	anuncio-aldi	TS	00:00:08	1280	720	25
5	Programa *Ahora Caigo*	ahora-caigo	TS	00:17:35	1280	720	25
6	Secuencia de anuncios	test1	TS	00:03:18	1280	720	25
7	Secuencia de anuncios	test2	TS	00:12:38	1280	720	25

Tabla 7: Características de los vídeos para pruebas.

Los **cuatro primeros** vídeos corresponden a pequeñas muestras de anuncios.

- **Anuncio de Booking** muestra un anuncio completo de Booking. En él, aparece el logotipo de la empresa pequeño durante gran parte del anuncio, y el logotipo en grande los segundos finales.
- **El anuncio de Aldi** consiste en el final de un anuncio, el inicio de un anuncio de la empresa Aldi, el final de este y el comienzo de otro anuncio.
- Los vídeos de **anuncios de Juver y Aldi** corresponden a las partes finales de anuncios de dichas empresas.

El **quinto vídeo** es una parte del programa de Antena 3 Ahora Caigo.

El **sexto vídeo** corresponde a la Secuencia de Anuncios utilizada para entrenamiento.

El **séptimo vídeo** es que tiene más duración, y será utilizado para medir los resultados del sistema.

4.4. DESCRIPCIÓN DEL SISTEMA

4.4.1. PROPÓSITO

El objetivo es crear un programa en Python que reciba: como argumentos una carpeta (contenedora de imágenes) y un vídeo; y como salida, el nombre del fichero y el intervalo de tiempo en el vídeo en el que aparece.

Por ejemplo:

En la ruta donde se encuentra el fichero .py, se introducirá la siguiente línea en la consola:

```
python programa.py datosX video_test.ts
```

Siendo:

1. **`python`**: argumento para iniciar un programa Python
2. **`programa.py`:** el nombre del programa Python que contiene el main
3. **`datosX`**: la carpeta que contiene las imágenes con los logotipos. Donde X será un número entero dependiente de la carpeta que se quiera seleccionar.
4. **`video_test.ts`:** el nombre del vídeo a analizar dentro de la carpeta datos.

En el caso de que la línea sea incorrecta y falten argumentos (o sobren) se mostrará un mensaje:

```
Debe introducir 4 parametros.
Ejemplo de ejecución:
>> python programa.py datosX test1.ts
```

Al ejecutar el programa, se irán mostrando las imágenes por orden de aparición. Se podrá determinar cuántos fotogramas por segundo se quieren analizar. Por defecto será 1 fotograma por segundo.

4.4.2. DIAGRAMA DE FLUJO.

4.4.2.1. Diagrama del programa general).

El siguiente diagrama muestra el funcionamiento general. En primer lugar, se leen los argumentos introducidos por la consola. En el caso de que el programa no encuentre estos archivos (bien sea porque no existen, o se ha introducido su nombre mal), la propia librería OpenCV genera un error por pantalla.

A cada imagen se le asignará un identificador. Como el procesado de las imágenes de cada fotograma para varias imágenes requería mucho tiempo de ejecución, se recurren a los procesos de Python para ejecutar varios hilos simultáneamente.

4.4.2.1.1. Diagrama de flujo en paralelo

Para ello se importa la librería:

```
import concurrent.futures
```

Para iniciar la función de búsqueda (explicada en la siguiente página), se ejecuta la siguiente línea de código:

```
with concurrent.futures.ThreadPoolExecutor(max_workers=20) as
executor:
            for num in fred:
                executor.submit(sift, num, seleccion1, seleccion2)
```

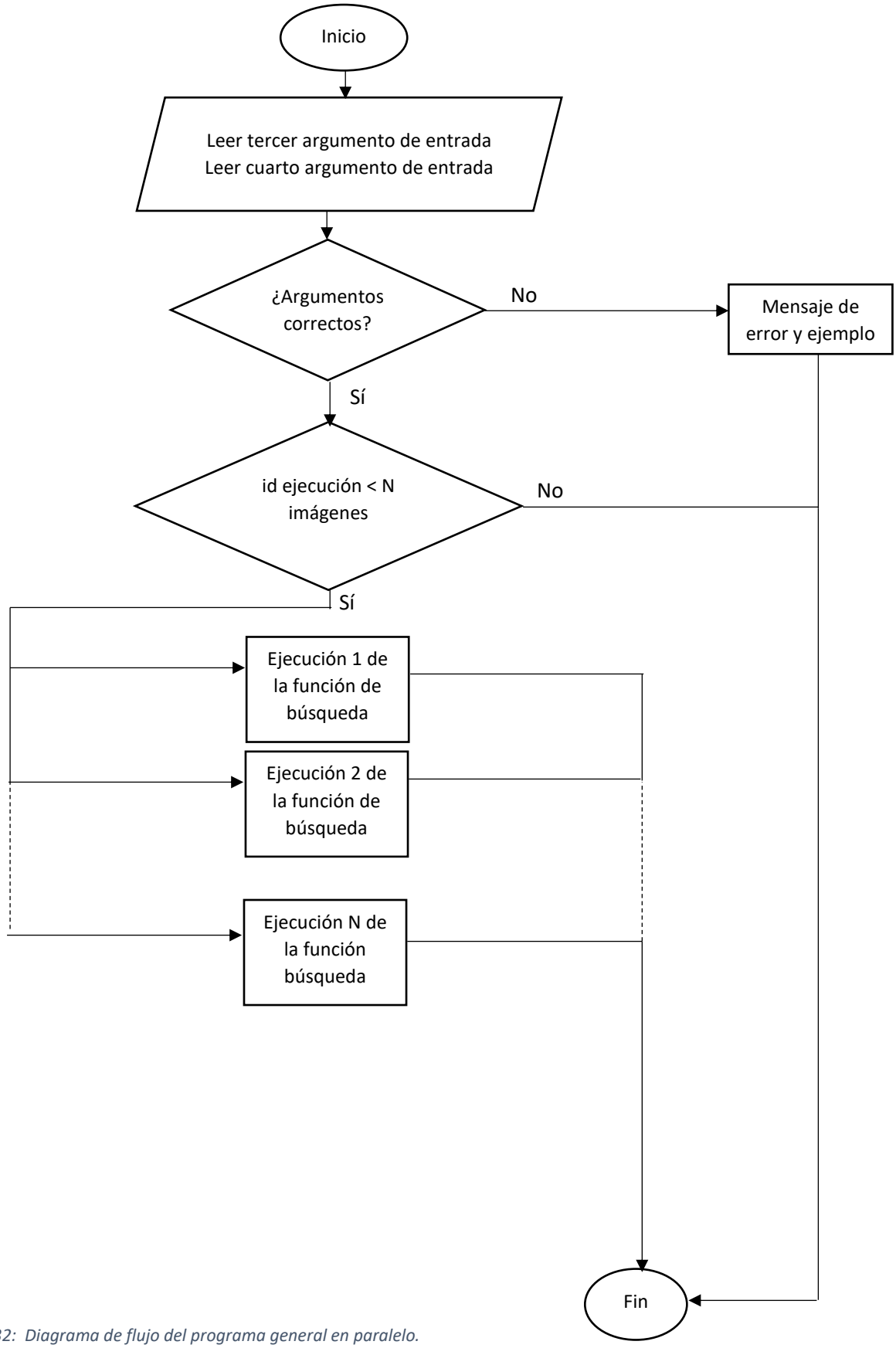


Figura 32: Diagrama de flujo del programa general en paralelo.

De esta forma, hemos creado 20 *workers* (como máximo) para iniciar la función de búsqueda (SIFT o SURF). Los argumentos para esta función será un identificador *num* del logo que se desea encontrar. En el diagrama este *num* será un número entero comprendido entre 1 y N, siendo N el número de logos a buscar. Por tanto, se ejecutarán tantos hilos como imágenes a analizar haya (sin contar los fotogramas del vídeo).

4.4.2.1.2. Diagrama de flujo en secuencial.

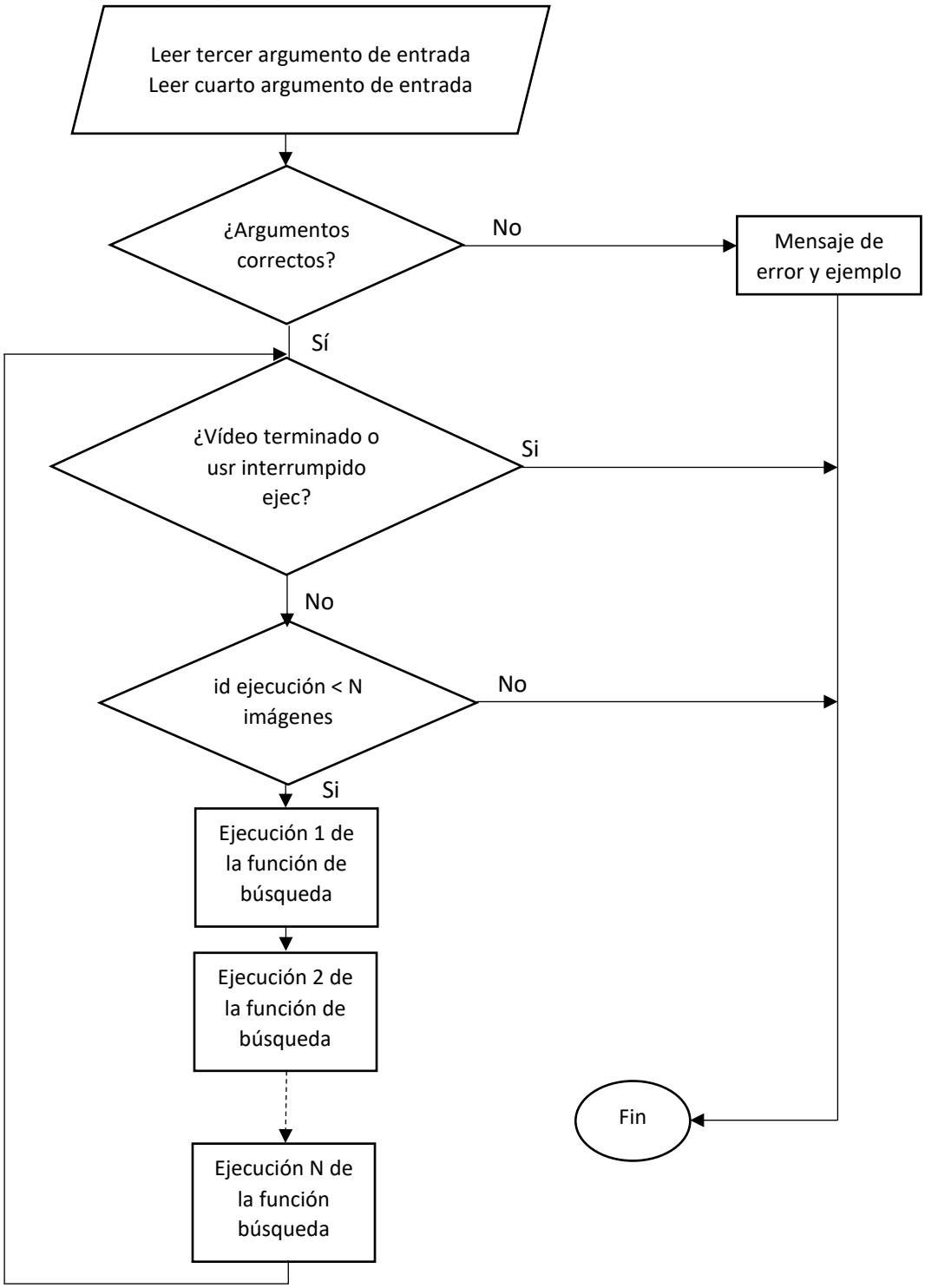


Figura 33: diagrama de flujo del programa general en secuencial.

El proceso también puede hacerse de forma secuencial. De esta forma, se analiza cada fotograma del vídeo con cada imagen a comparar una por una (Ejecución X de la función de búsqueda una por una). Esta forma de ejecución puede ser útil para representan por pantalla el número de matches de todos los logos en una misma ventana.

4.4.2.2. Diagrama de la función de búsqueda

Los diagramas de la Figura 34 y 35 corresponden al funcionamiento de la función de búsqueda, en paralelo y en secuencial, respectivamente.

Con el fotograma del vídeo y el identificador de la imagen del logo, se obtienen las dos imágenes a comparar. Por defecto, el programa analiza un fotograma cada segundo, de los 25 posibles. En primer lugar, se convierten a escala de grises, ya que la máquina identificará la intensidad de los píxeles de su alrededor, y trabaja mejor de este modo.

Para comprender mejor los pasos *Escoger detector* y *Escoger y configurar matcher* se recomienda leer la sección 2.4 de esta memoria. En la sección 2.4 se explica el funcionamiento del algoritmo SIFT y SURF, cómo se crean los descriptores y cuál es la función de OpenCV que lo permite realiza en Python. Se explica cómo funcionan los emparejadores (*matchers*) por fuerza bruta BFMatchers, y los basados en FLANN.

El procesado del fotograma se refiere al conjunto de procesos por los que pasa el fotograma para mejorar la imagen, como puede ser la eliminación de ruido y la ecualización de la imagen.

A continuación, se cuenta el número de matches entre cada fotograma del vídeo, y cada uno de los logos a encontrar. Si el número de coincidencias supera un determinado umbral, se analizan estos puntos por RANSAC (explicado en la sección 2.4.10), donde deberá superar un segundo umbral. En este caso, la detección se considera válida y se activa un flag de detección para este anuncio. Además, se generan unos archivos de salida con tres columnas para cada imagen, en las que se imprime: los matches, los matches correctos, y el instante de tiempo cuando se han producido.

Paralelamente a este proceso, se está comprobando fotograma por fotograma (si el vídeo es de 25 fps, se analizarán los 25), si la imagen por pantalla es completamente en negro. Esto indicará las transiciones entre anuncios. Para ello, se contará el número de píxeles que no son negros. Para ello se utilizará la función `cv2.countNonZero(img1) < 8000`. Donde 8000 es un umbral que se ha establecido a través de pruebas. Al tratarse de un vídeo, muchos píxeles de, por ejemplo, 1280 x 720 = 921 000, no serán completamente negros, aunque la transición exista.

Cuando se detecta esta transición entre anuncios, se pone un contador específico a cero, que indica que un nuevo anuncio se está ejecutando. Cuando se vuelve a detectar un fotograma en negro, se comprueba si existe algún flag de detección para ese anuncio. De ser así, se imprime el resultado por pantalla, se vuelve a poner el contador interno de los anuncios a cero y se repite el proceso.

4.4.2.2.1. Diagrama de la función de búsqueda en paralelo.

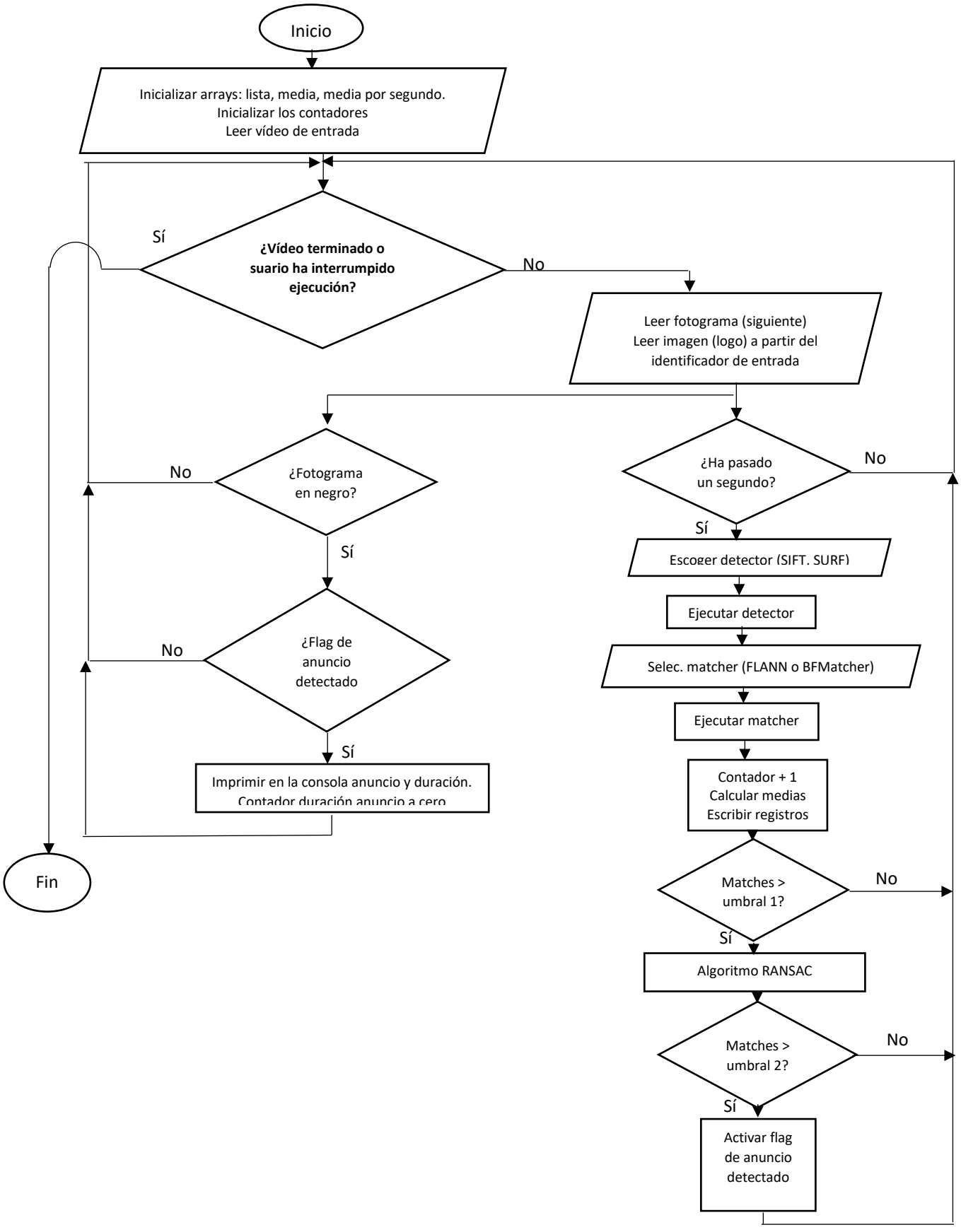


Figura 34: Diagrama de flujo del algoritmo de la función de búsqueda en paralelo.

4.4.2.2.2. Diagrama de la función de búsqueda en secuencial.

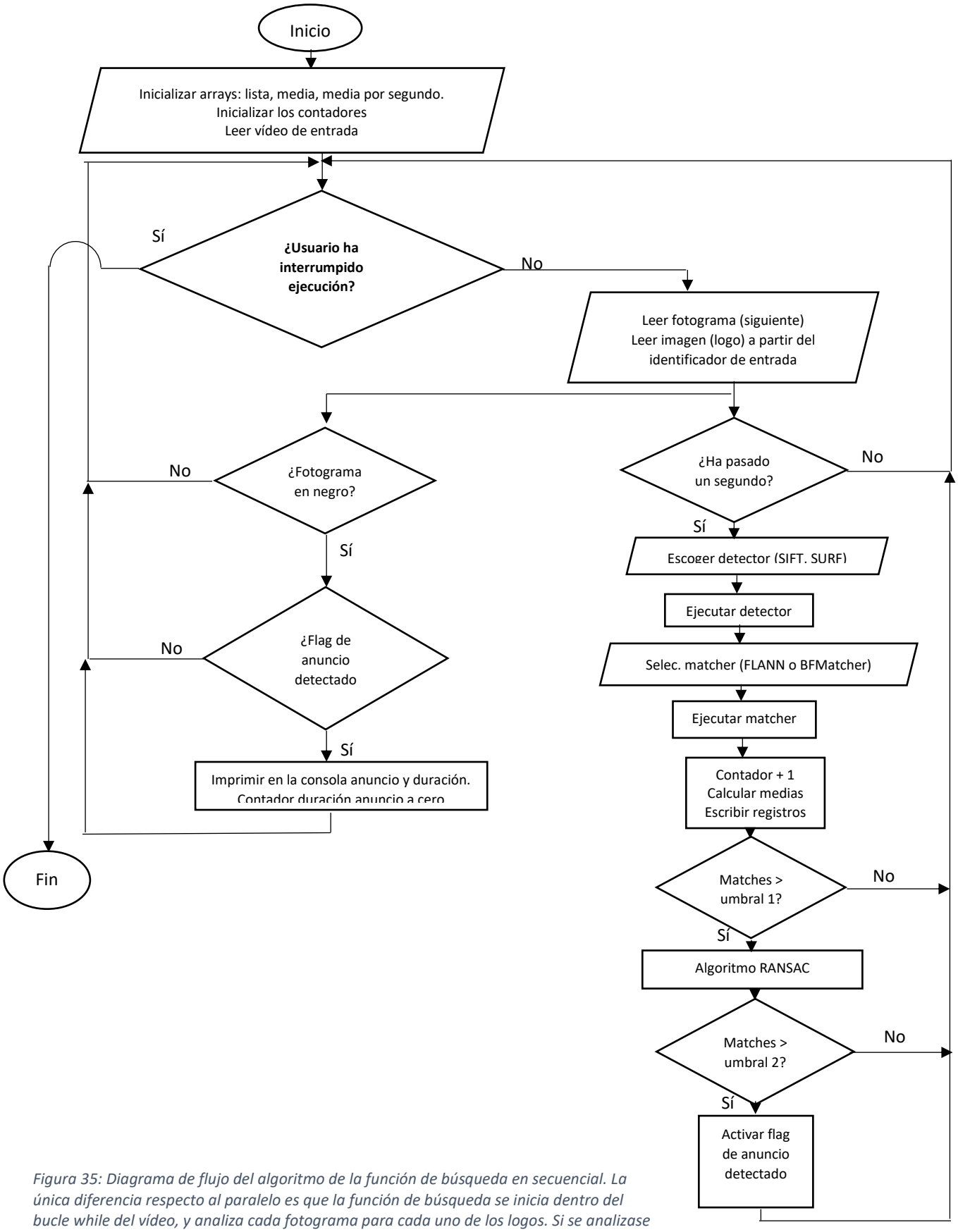


Figura 35: Diagrama de flujo del algoritmo de la función de búsqueda en secuencial. La única diferencia respecto al paralelo es que la función de búsqueda se inicia dentro del bucle while del vídeo, y analiza cada fotograma para cada uno de los logos. Si se analizase el vídeo completo para cada muestra, no se podría soportar una función de streaming.

Existe la opción para analizar más fotogramas por segundo. En este caso, se analizarán 5 ó 25 fotogramas por segundo. Se realizará la media de cada segundo, de la forma:

$$\mu_i^{(j)} = \frac{1}{m}\sum_{i\in N}^{N} x_i^{(j)}$$

Donde:

- **m:** número de fotogramas en un segundo. Puede ser 5 o 25.
- **i:** número de fotograma, de 1 a hasta N. Si se quiere analizar un vídeo de 25 fps, y m = 5, entonces se analizará el fotograma i = 1, 6, 11, 16, 21.
- **j**: identificador del anuncio.

Análisis para anuncio j:

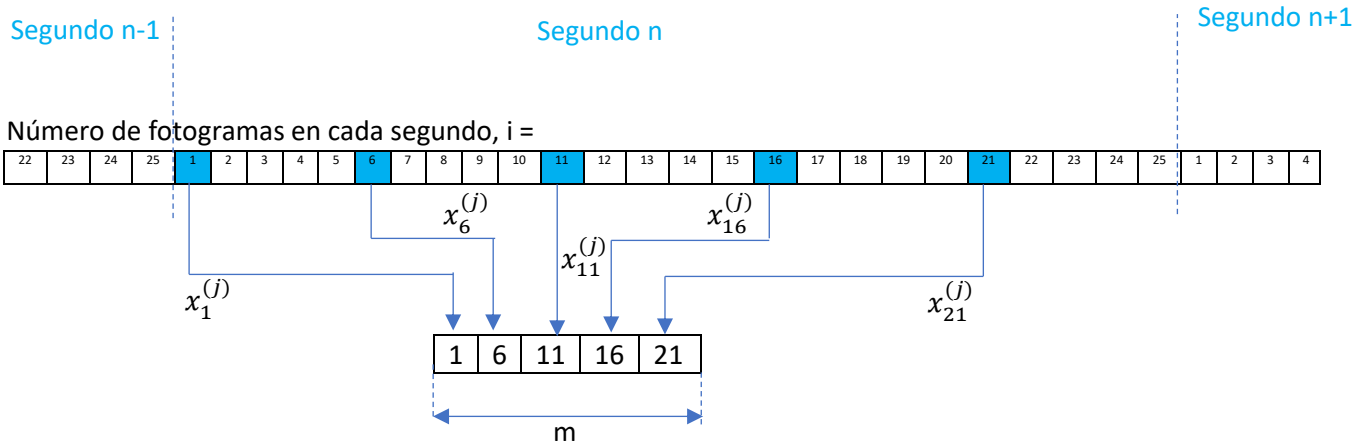


Figura 36: Funcionamiento de las medias de matches.

Entre las funciones auxiliares, se encuentra un conversor de segundos en entero a formato hh:mm:ss; y una función cargar_carpeta, que carga todas las imágenes que contiene una carpeta que se pasa por parámetro.

El programa finaliza cuando se acaba el vídeo.

4.4.3.ÁRBOL

La carpeta principal del programa contendrá un fichero y cuatro carpetas:

- El fichero es el programa .py a ejecutar.
- La carpeta datos contiene los vídeos con las secuencias de imagen
- La carpeta **datosX** contienen los ficheros de las imágenes de los logos que se quieran buscar. Si esta carpeta contiene otra carpeta dentro con logos, se detectarán los logotipos de su interior, pero en la consola de la salida aparecerá solo el nombre de la carpeta. Por lo que se recomienda que solo contenga los ficheros de imagen.
- La **carpeta reg** contendrá unos ficheros .txt para cada logo. Su nombre empezará por registro_, seguido del nombre del fichero al que pertenece (con su extensión). Estos ficheros contendrán tres columnas. La primera contiene el número de matches en total, la segunda los matches reales, y la tercera columna contendrá el tiempo donde se han encontrado. El valor de estos intervalos dependerá del número de fotogramas que se analizan cada segundo.
- **La carpeta registro_final** contiene también unos ficheros del mismo formato que la carpeta reg, pero el resultado será la media de matches de cada segundo. Si solo se analiza un fotograma por segundo, los ficheros de reg y registro_final serán iguales.

La organización de la aplicación es la siguiente:

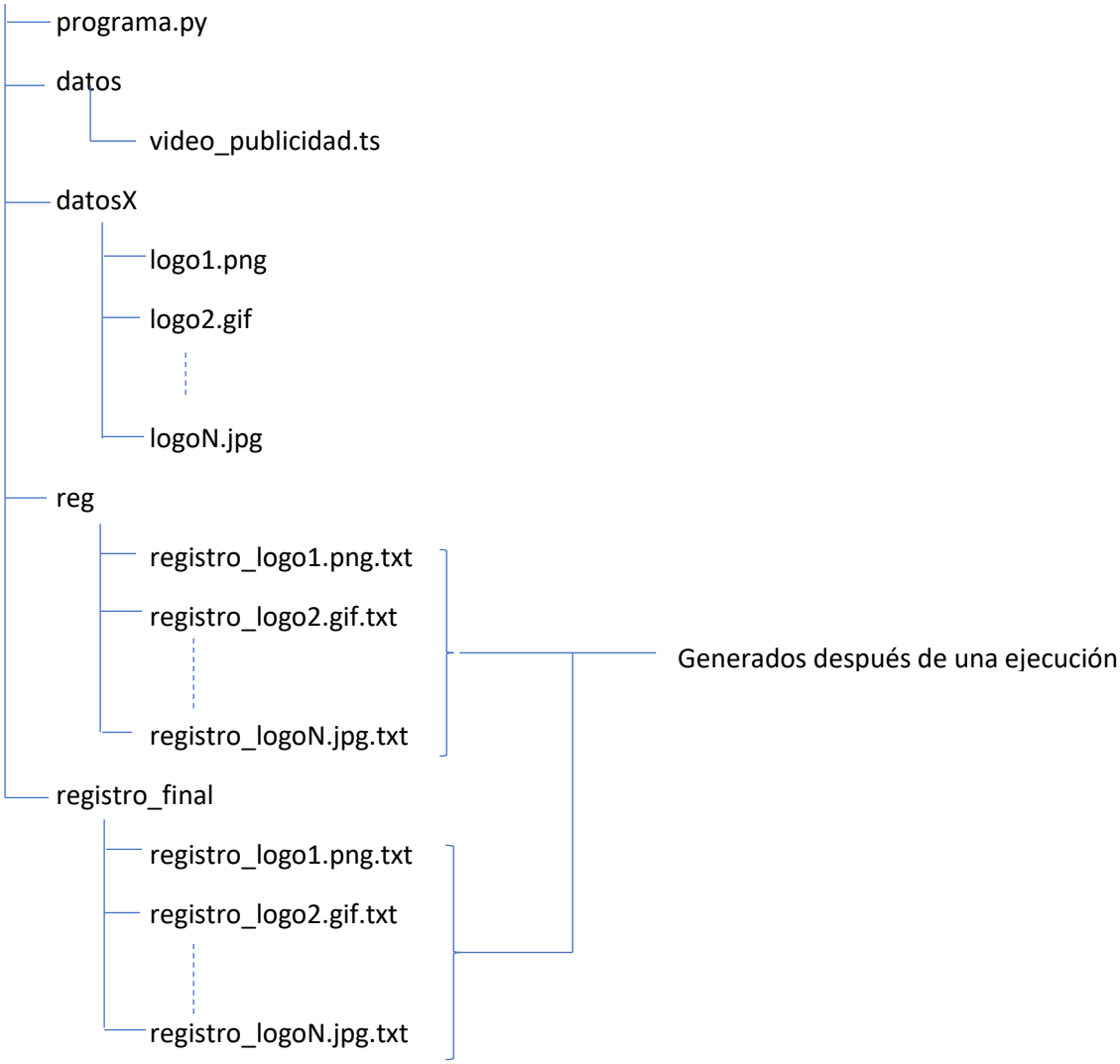


Figura 37: Árbol de la estructura del programa.

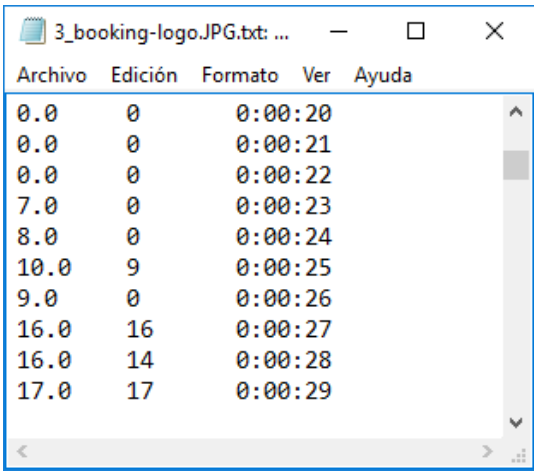


Figura 38: Ejemplo de un archivo de la carpeta registro-final. El registro tiene formato .txt y se corresponde con la imagen 3_booking-logo.JPG. En la primera columna, los matches totales, en la segunda, los matches correctos, y en la tercera, el instante de tiempo cuando se producen.

5. RESULTADOS OBTENIDOS

Esta sección constará de dos partes. En la primera se analizará un vídeo y logos de entrenamiento para establecer algunos valores del algoritmo. En la segunda parte, se medirá la efectividad del programa para detectar anuncios en otros vídeos.

5.1. DATOS DE ENTRENAMIENTO

En este apartado se entrenará nuestro algoritmo para determinar los atributos del mismo, y el umbral a tener en cuenta. Esto se hará midiendo error (la no detección o detección incorrecta), y seleccionando los parámetros para los cuales este error sea mínimo.

5.1.1 FEATURE MATCHING: DETECCIÓN DE LA MOSCA

5.1.1.1. Canales

Se van a analizar los 5 canales de televisión con más audiencia en España.

LA1	LA2	ANTENA3	CUATRO	TELECINCO
				

Tabla 8: logotipos de las cadenas de televisión.

#	CANAL	NOMBRE FICHERO	FORMATO	TAMAÑO	DIMENSIONES
1	La 1	logo-la1	PNG	771 KB	1200x1024
2	La 2	logo-la2	PNG	400 KB	1200x1069
3	Antena 3	logo-a3-2	JPG	15 KB	332x274
4	Cuatro	logo-cuatro	PNG	6.18 KB	500x500
5	Telecinco	logo-t5	PNG	10.2 KB	500x500

Tabla 9: Características de las imágenes.

5.1.1.2. Resultados

Para detectar el logotipo del canal que se está emitiendo, nos aprovecharemos del hecho de que esta imagen suele aparece en una de las esquinas de la pantalla. Esta posición varía según la cadena. Para La 1, La 2, Antena 3 y Telecinco suele ser la esquina inferior derecha. Para cuatro, la superior izquierda.

Para seleccionar los píxeles determinados se escribe en el código:

```
imagen2 = img[900:1200, 400:660]
```

Donde Imagen2 será la porción definida entre corchetes de la imagen img.

Vídeo: programa *Ahora Caigo* (Antena 3)	ahora-caigo.ts

Logos a comparar: Antena 3, La 1 y Booking

EMPRESA	MEDIA *MATCHES*	DECISIÓN
ANTENA3	1.0	1
LA 1	0.0	0
BOOKING	0.5	0

Tabla 10: Detección de la mosca.

Figura 39: Sección escogida para analizar el logotipo.

5.1.2. FEATURE MATCHING: DETECCIÓN DE LOGOS EN ANUNCIOS

5.1.2.1. Recursos

#	EMPRESA	NOMBRE FICHERO	FORMATO	TAMAÑO	DIMENSIONES
1	Aldi	aldi_logo2_pequeno	JPG	14.7 KB	347x151
2	Balearia	Balearia_video	PNG	49.5 KB	472x97
3	Booking	3_booking-logo	JPG	14.6 KB	377x112
4	Danone	1_danone_video	JPG	8.66 KB	78x37
5	Flex	logo_flex_video	PNG	69.1 KB	525x94
6	Juver	2_logo_juver	PNG	28.8 KB	567 x 283
7	La Tienda en casa	Ltc_video3	PNG	39.7 KB	257x195
8	Rexona	rexona_video	PNG	67.7 KB	289x299

Tabla 11: características de las imágenes utilizadas para la prueba del vídeo test1.ts

Aldi	Balearia	Booking	Danone
ALDI SUPERMERCADOS	BALEARIA	Booking	DANONE
Flex	**JUVER**	**LA TIENDA EN CASA**	**REXONA**
FLEX	Juver	LTC	Rexona

Tabla 12: Imágenes de los logotipos de las empresas a detectar.

5.1.2.2. El Logotipo no se encuentra en primer plano

Vídeo: anuncio de Booking

Hay anuncios en los que el logo aparecerá en pequeño, y estará rodeado de otros elementos. El algoritmo podrá localizarlo y establecer una serie de *matches,* concentrados en el logo.

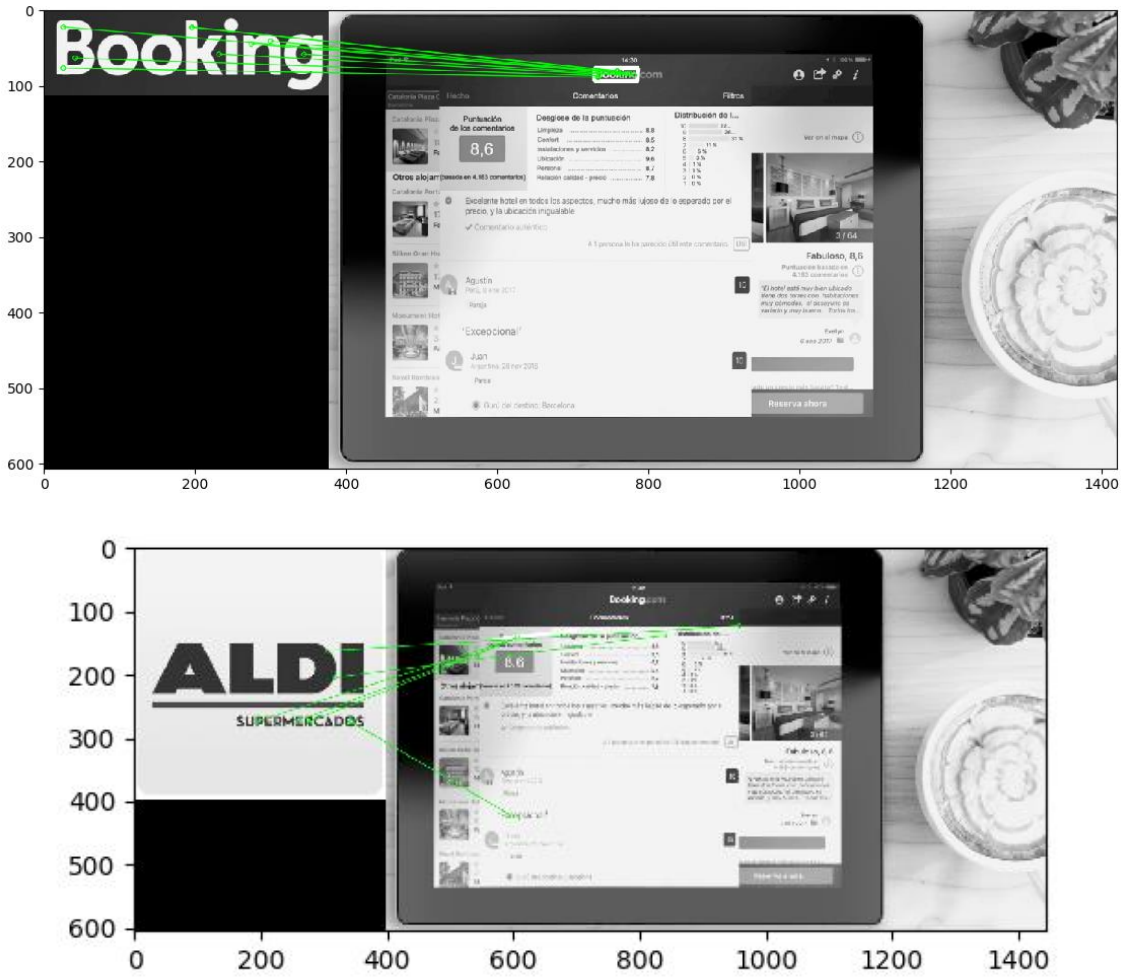


Figura 40: Ejemplos de la muestra por pantalla sin el anuncio en primer plano. En el primer ejemplo se compara el logo de Booking con un fotograma de un anuncio de Booking que lo contiene. En el segundo ejemplo se compara un logo de Aldi con el mismo vídeo.

En esta prueba, se ha observado que, cuando los matches son correctos, eso se concentran en una pequeña región de la imagen (que contiene la imagen), y a medida que pasan los segundos, el número de matches no muestra apenas varianza. En cambio, cuando los matches son incorrectos suelen aparecer dispersos, y el número de matches cambia cada segundo y muestra una alta varianza.

A continuación, se representa los primeros segundos de la ejecución de una prueba para detectar un logo en un fotograma.

Booking	Aldi
Enough matches are found - 47/10 Booking Enough matches are found - 40/10 Booking Enough matches are found - 41/10 Booking Enough matches are found - 45/10 Booking Enough matches are found - 47/10 Booking Enough matches are found - 49/10 Booking Enough matches are found - 50/10 Booking Enough matches are found - 48/10 Booking Enough matches are found - 44/10 Booking Enough matches are found - 43/10 Booking Enough matches are found - 46/10 Booking	Enough matches are found - 63/10 Aldi Enough matches are found - 60/10 Aldi Enough matches are found - 64/10 Aldi Enough matches are found - 56/10 Aldi Enough matches are found - 58/10 Aldi Enough matches are found - 49/10 Aldi Enough matches are found - 43/10 Aldi Enough matches are found - 35/10 Aldi Enough matches are found - 28/10 Aldi Enough matches are found - 64/10

Tabla 13: Ejemplo de ejecución en la consola. Primero, se muestra una frase si indica si se ha superado cierto umbral o no. Después el número que índica el número de matches detectados. Seguido de la barra, se muestra un umbral para determinar si una imagen ha sido detectada o no. En este caso, por defecto se muestra 10, aunque se cambiará más adelante.

5.1.2.3. Con logo en primer plano

En la mayoría de los anuncios, la parte metraje más adecuada para clasificar un anuncio se encuentra al final de él, durante los últimos segundos finales. En esta parte, se muestra el logotipo en grande, con el fondo característico, y a menudo el lema de la empresa. Este será el mejor momento para poder detectar el anuncio.

Vídeo de anuncio de Juver	anuncio-juver.ts

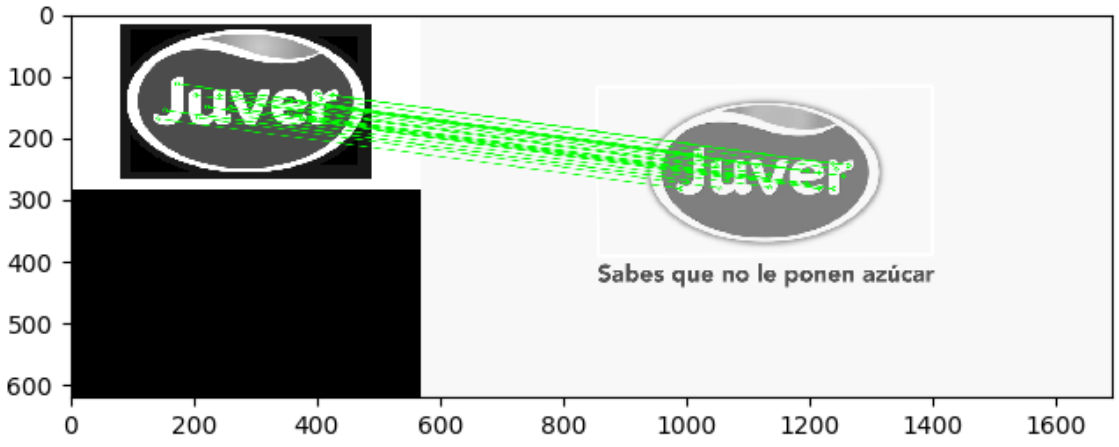


Figura 41: Detección de Juver. Logotipo en primer plano.

5.1.3. CONCLUSIONES DE LAS DETECCIONES EN UN SOLO ANUNCIO

- La velocidad es muy baja: se analizará un fotograma por segundo
- Algunas imágenes dan valores muy dispersos: se cambiará el factor de distancia entre el primer y segundo mejor resultado. (Ver sección: SIFT)
- Se aumentará el número de árboles. (Ver sección: SIFT)

5.1.4. ANÁLISIS PARA UNA CADENA DE ANUNCIOS

5.1.4.1. SIFT + FLANN

5.1.4.1.1 Recursos

Vídeo:	test1.ts (fps = 25)

Tabla 14: vídeo utilizado en la Prueba del vídeo test1

5.1.4.1.2. Análisis de anuncios

Tiempo	**Predicción**	**Conteo real**
0:00:02 - 0:00:22	1_danone_video.JPG	Danone
0:00:22 - 0:00:42	3_booking-logo.JPG	Booking
0:00:42 - 0:01:03	aldi-logo_v.PNG	Aldi
0:01:03 - 0:01:18	-	Promo Antena 3
0:01:18 - 0:01:38	2_logo_juver.png	Juver
0:01:38 - 0:01:59	-	Promo Antena 3
0:01:50 - 0:02:19	logo_flex_video.PNG	Flex
0:02:19 - 0:02:39	ltc_video3.PNG	La tienda en Casa
0:02:39 - 0:02:49	rexona_video.PNG	Rexona
0:02:49 - 0:03:09	balearia_video.PNG	Balearia

Tabla 15: comparación de los resultados obtenidos mediante programa, y conteo real

5.1.4.1.3. Matches en función del tiempo

Fotograma en negro en (eje de tiempo): 2, 22, 42, 63, 78, 98, 118, 139, 159, 169, 189.

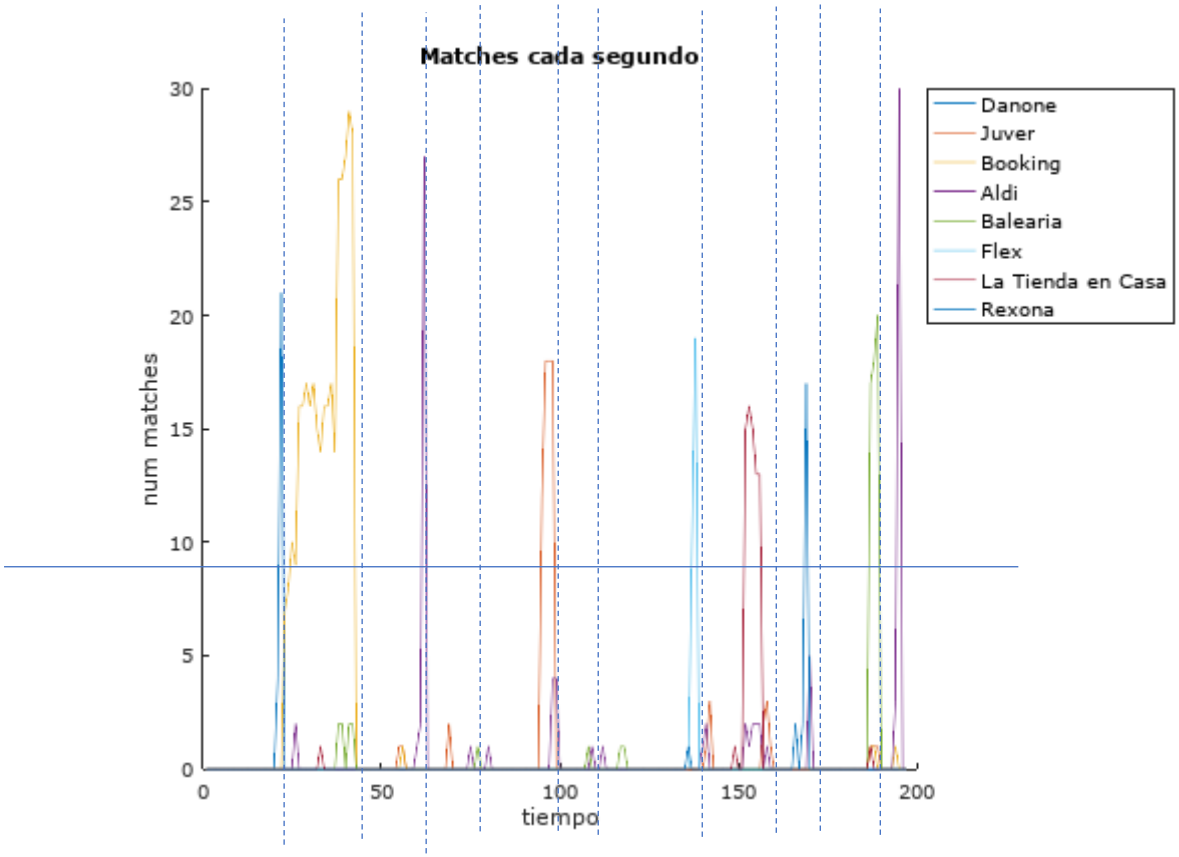


Figura 42: Representación del número de coincidencias respecto al tiempo. Aunque el último segmento está incompleto y no se analiza en esta prueba, se produce una detección incorrecta, ya que se detecta Aldi. Estos errores son tratados más adelante.

5.1.4.1.4. Matriz de confusión

A continuación, se muestra una tabla de Confusión donde se analizan los matches. El programa emite su «veredicto» una vez el anuncio ha acabado. En este momento lee un flag que podrá haber sido activado o no. Este flag se activa cuando el número de matches de una imagen supera cierto umbral. Para ello se analizará para segmento del vídeo. Estos segmentos contendrán cada anuncio o promoción. Se pondrá el número de matches máximo de cada imagen en cada segmento, para determinar un umbral.

		VALOR PREDICHO							
		Danone	Booking	Aldi	Juver	Flex	LTC	Rexona	Balearia
VALOR REAL	Danone	21	0	0	0	0	0	0	0
	Booking	0	29	2	0	0	1	0	2
	Aldi	0	1	34	1	0	0	0	0
	Juver	0	0	4	18	0	0	0	0
	Flex	0	0	1	0	19	0	1	1
	LTC	0	0	2	3	0	16	0	0
	Rexona	0	0	0	0	0	0	17	0
	Balearia	0	1	5	1	0	1	0	20

Tabla 16: Matriz de confusión con SIFT. En matches totales.

En porcentaje de aciertos (%):

		VALOR PREDICHO							
		Danone	Booking	Aldi	Juver	Flex	LTC	Rexona	Balearia
VALOR REAL	Danone	100	0	0	0	0	0	0	0
	Booking	0	85.29	5.88	0	0	2.94	0	5.88
	Aldi	0	2.78	94.44	2.78	0	0	0	0
	Juver	0	0	18.18	81.81	0	0	0	0
	Flex	0	0	4.54	0	86.36	0	4.54	4.54
	LTC	0	0	9.52	14.28	0	76.19	0	0
	Rexona	0	0	0	0	0	0	100	0
	Balearia	0	3.56	17.85	3.56	0	3.56	0	71.43

Tabla 17: Matriz de confusión con SIFT. En porcentajes.

5.1.4.2. SURF + FLANN

Umbral hessiano del objeto SURF en 400. Umbral para detección: 12 matches.

5.1.4.2.1. Análisis de anuncios

Tiempo	**Predicción**	**Conteo real**
0:00:02 - 0:00:22	-	Danone
0:00:22 - 0:00:42	3_booking-logo.JPG	Booking
0:00:42 - 0:01:03	aldi-logo_v.PNG	Aldi
0:01:18 - 0:01:38	2_logo_juver.png	Juver
0:02:19 - 0:02:39	ltc_video3.PNG	La tienda en Casa
0:02:39 - 0:02:49	rexona_video.PNG	Rexona
0:02:49 - 0:03:09	balearia_video.PNG	Balearia

Tabla 18: Ejecución con detector y descriptores de SURF, y Matcher FLANN

5.1.4.2.2. Matches en función del tiempo

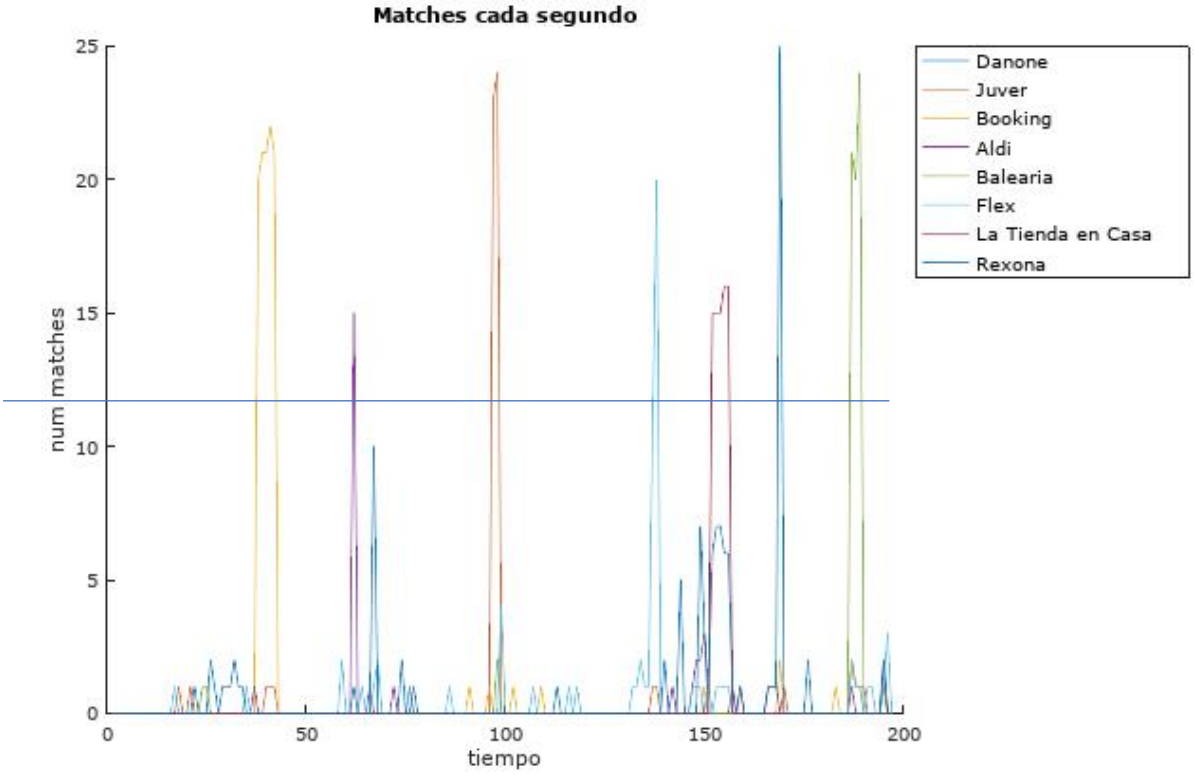


Figura 43: En la gráfica se muestra, como los anuncios son detectados puesto que superan un umbral establecido. El logo de danone no se detecta en SURF, debido a que esta configuración tan solo detecta 1 match en el minuto 00:22.

5.1.4.2.3 Matriz de confusión.

Se mide el número máximo de matches para cada imagen (valor predicho) en cada segmento del vídeo.

		Valor predicho							
		Danone	Booking	Aldi	Juver	Flex	LTC	Rexona	Balearia
Valor Real	Danone	1	0	0	0	0	0	0	0
	Booking	0	22	0	1	0	0	2	0
	Aldi	0	1	15	0	0	0	1	0
	Juver	1	1	0	24	0	1	1	0
	Flex	1	1	2	1	16	1	0	0
	LTC	0	0	0	0	0	16	0	0
	Rexona	0	2	1	0	0	6	25	0
	Balearia	0	1	0	1	0	0	0	25

Tabla 19: Matriz de confusión. SURF y FLANN

5.1.4.3. Matches incorrectos y solución

Existen imágenes, que debido a sus características generan matches cuando corresponde, pero también en partes del vídeo donde no, llegando incluso a superar el umbral de detección. Se producen, por tanto, detecciones incorrectas.

Para solucionar este problema, y seleccionar solo lo matches correctos, se puede RANSAC mediante la función findHomography, descrita en la sección 2.4,10.

En la Figura 42 se muestra una gráfica de los matches totales y los correctos en función del tiempo. La figura es la misma en los dos casos, y corresponde al logotipo de Aldi que no se ha utilizado como muestra final.

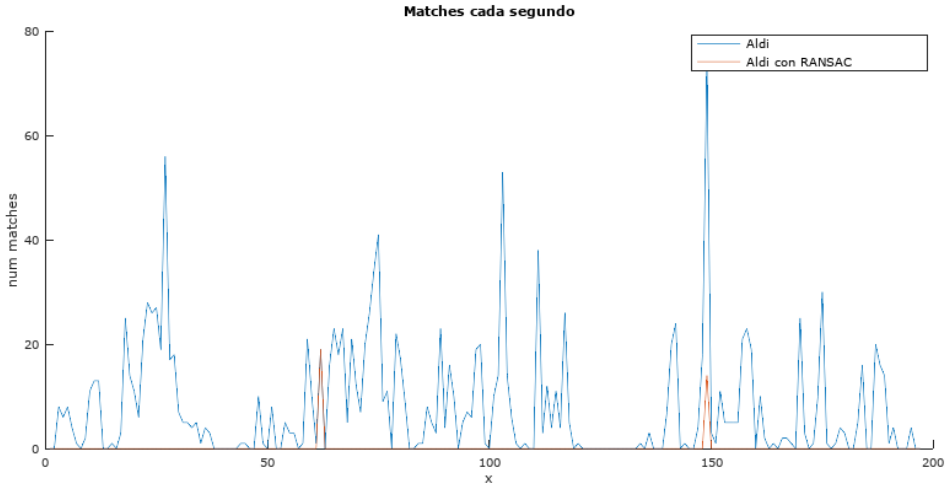


Figura 44: Se muestra el número de matches de la misma imagen. En la muestra azul, solo se ha aplicado SURF. Es una mala muestra, ya que supera el umbral en varios puntos en los que no corresponde. En rojo, se marca los matches con el algoritmo RANSAC. Se producen 19 matches en el punto del tiempo correcto, y otro pico erróneo, pero que no supera el umbral para tenerlo en cuenta. Se ha reducido mucho la probabilidad de error.

5.1.4.4. SURF + FLANN con RANSAC

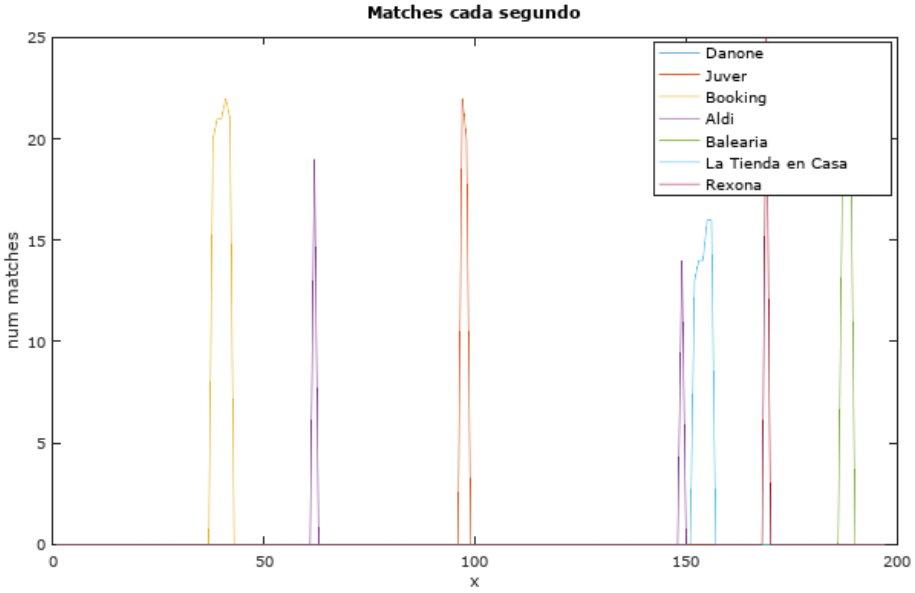


Figura 45: Se aplica RANSAC para las muestras que superen cierto umbral. El número de muestras detectadas y correctas se imprimen en una tercera columna (entre los matches sin RANSAC y el tiempo). Para el resto de matches, se asigna valor 0.

		Valor predicho							
		Danone	Booking	Aldi	Juver	Flex	LTC	Rexona	Balearia
Valor Real	Danone	1	0	0	0	0	0	0	0
	Booking	0	22	0	0	0	0	0	0
	Aldi	0	0	19	0	0	0	0	0
	Juver	0	0	0	22	0	0	0	0
	Flex	0	0	0	0	14	0	0	0
	LTC	0	0	2	0	0	16	0	0
	Rexona	0	2	0	0	0	0	25	0
	Balearia	0	0	0	0	0	0	0	20

Tabla 20: valores con RANSAC (se utiliza la imagen, a priori con errores, de Aldi) BFMAtcher

5.1.5. EXPLICACIÓN DEL SISTEMA COMPLETO. SIFT, BFMATCHER Y RANSAC.

Fotograma en negro en (eje de tiempo): 2, 22, 42, 63, 78, 98, 118, 139, 159, 169, 189.
Representados con línea de puntos azul: ----------

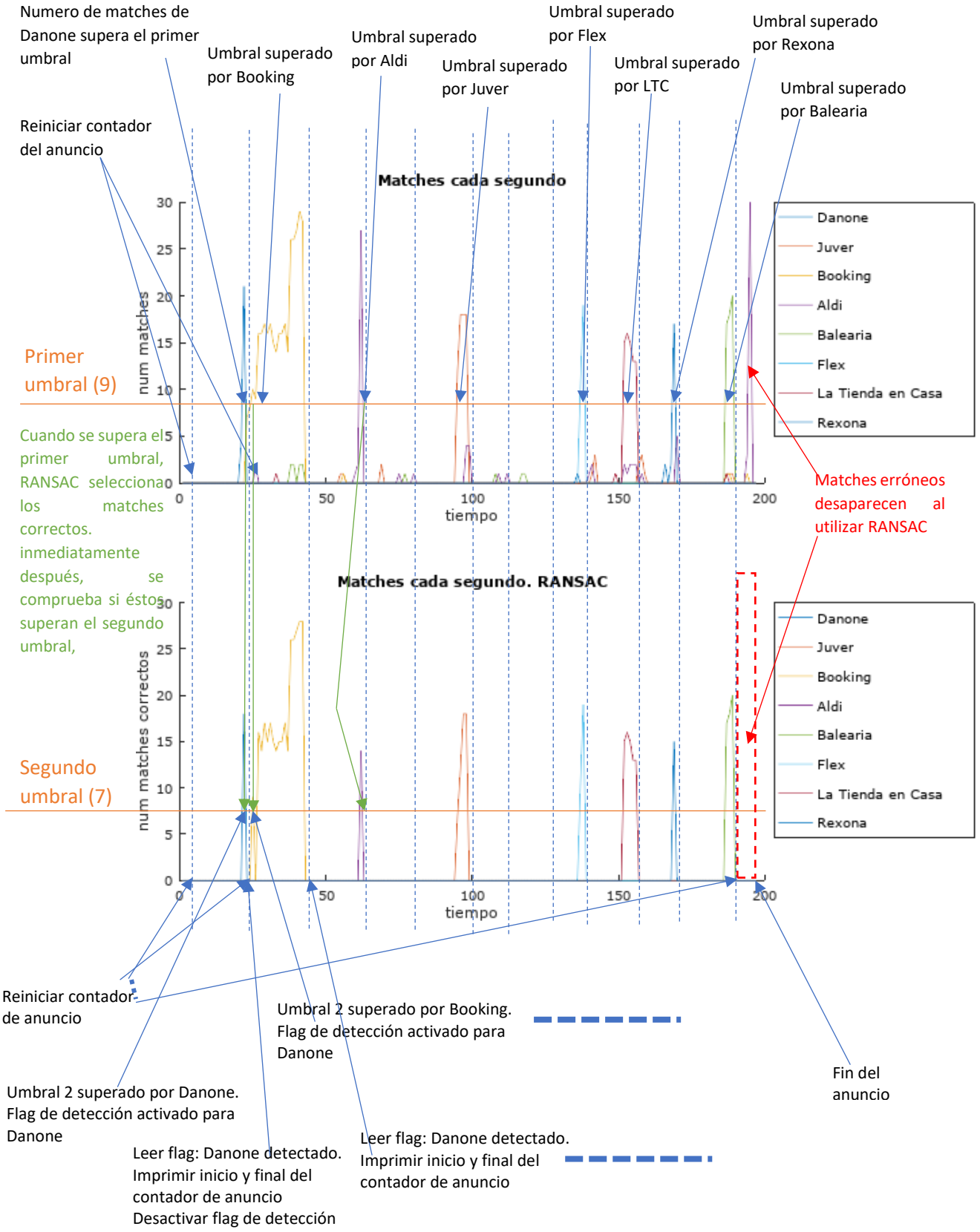


Figura 46: Funcionamiento del sistema completo para las muestras de entrenamiento. Algoritmo con Detector SIFT, matcher de Fuerza Bruta, y algoritmo RANSAC.

5.1.6. RENDIMIENTO

Comparación de la ejecución del código analizando cada logotipo en el vídeo de forma secuencia y en paralelo.

5.1.6.1. Computación secuencial.

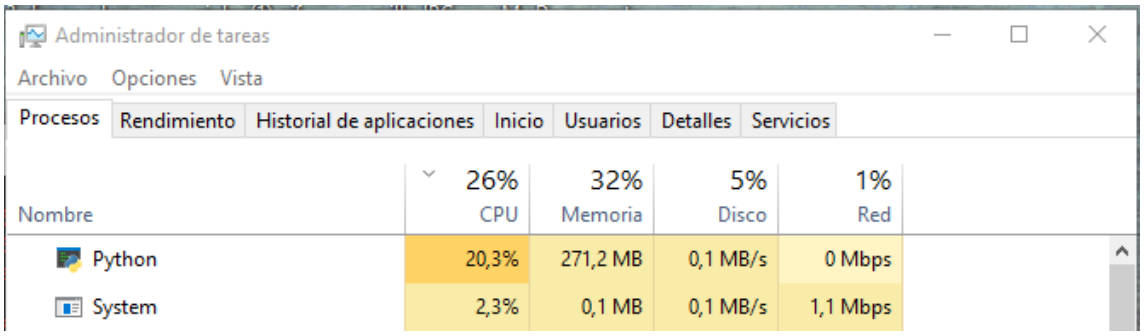


Figura 47: Rendimiento del ordenador cuando la ejecución es secuencia. Prestar valor al uso de la CPU por parte del programa Python.

En este caso, primero se extrae el fotograma de un vídeo, y se compara uno por uno con el resto de logos.

5.1.6.2. Computación en paralelo

Se analizará cada imagen en diferentes hilos. De esta forma se utiliza toda la capacidad del procesador, se analizan todos los logos casi a la vez, de forma que se imprimen por pantalla según su capacidad (algunas imágenes se comparan más lento que otras). Si hubiese un error en alguna imagen, la otras podrían procesarse independientemente. Además, se reduce el tiempo de ejecución (apartado 5.1.6).

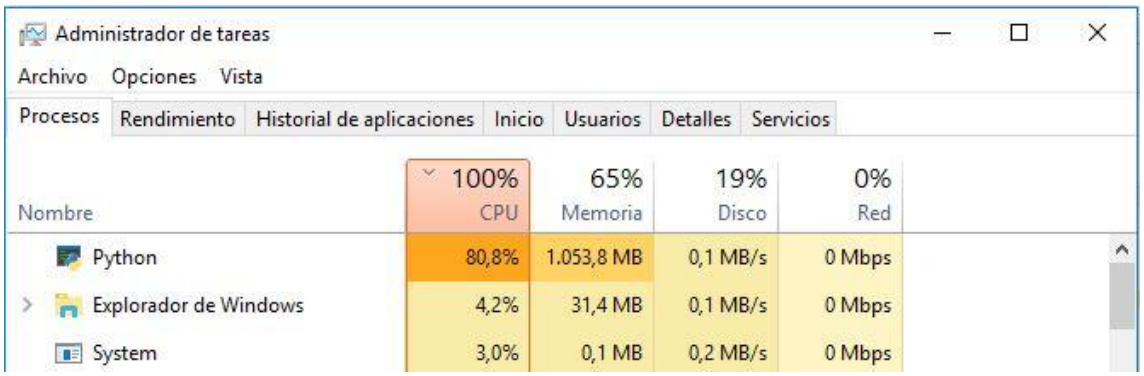


Figura 48: Rendimiento del ordenador con la ejecución de subprocesos en paralelo.

5.1.7. CONCLUSIONES DE LAS PRUEBAS

Para las muestras escogidas, se ha obtenido mejores resultados para SIFT que para SURF. El segundo algoritmo no detectaba de forma correcta un logo. Se ha comprobado que con los *matchers* BFMatcher y FLANN, se obtienen resultados muy similares con ambos. Se ha solucionado el problema de las detecciones falsas con el algoritmo RANSAC.

Parámetros escogidos:

- **Primer umbral** de detección: se fija en 9 matches.
- **El segundo umbral**, para los matches después de RANSAC: se fija en 7 matches.

Es decir, para que una imagen se considere detectada se tienen que encontrar un mínimo de 9 coincidencias, y de esas coincidencias, un mínimo de 7 deben ser una continuidad para considerase correctas.

- **Para SURF, el umbral Hessiano**: se fija en 200. Un valor superior genera muy pocas coincidencias, y uno inferior, demasiadas.
- Para el **matcher FLANN**. Se fijará trees = 20, y checks = 100, para obtener mayor precisión, ya que, según las pruebas, afectaba a la velocidad, pero no demasiado.
- La **relación de distancia** entre la primera mejor coincidencia y la segunda mejor se fija en 0.3. Un número mayor (e.g 8) provocaba demasiadas coincidencias.

5.2. DATOS DE TEST: SIFT Y BFMATCHER

Se escoge SIFT que es el algoritmo que mejores resultados ha dado, y el matcher BFMatcher.

5.2.1. RECURSOS

5.2.1.1. Vídeo

NOMBRE	FOTOGRAMAS POR SEGUNDO
TEST2.TS	25

Tabla 21: Características del vídeo de test SIFT y BFMatcher.

5.2.1.2. Imágenes

BBVA	**Burger King**	**Dulcolaxo**	**Fontbella**
			
Jazztel	**Mutua madrileña**	**Once**	**Schweppes**
			
Trivago	**Vodafone**	Arriaga Asociados	Carglass
			
Dormidina	Nocilla	Pantene	Tampax
			

Tabla 22: Imágenes de los logos utilizados en el test con SIFT y BFMatcher.

#	EMPRESA	NOMBRE FICHERO	FORMATO	TAMAÑO	DIMENSIONES
1	BBVA	bbva1	PNG	20.6 KB	217x83
2	Burger King	burger12	PNG	120 KB	292x285
3	Dulcolaxo	Dulcolaxo1	PNG	35 KB	450x88
4	Fontbella	font0	PNG	68.7 KB	410x233
5	Jazztel	jazz0	PNG	5,58 KB	248x76
6	Mutua Madrileña	mutua0	JPG	44.8 KB	500x176
7	Once	once1	PNG	15.1 KB	148x38
8	Schweppes	schwe0	PNG	105 KB	348x317
9	Trivago	trivago20	PNG	12,0 KB	470x166
10	Vodafone	vodafone-logo2	JPG	9,31 KB	188x188
11	Arriaga Asociados	arriaga	JPG	63.7 KB	900x900
12	Carglass	carglass2	PNG	23.7 KB	559x237
13	Dormidina	dormidina-logo3	PNG	17.5 KB	159x52
14	Nocilla	nocilla02	PNG	150 KB	529x310
15	Pantene	pantene1	PNG	16.6 KB	155x106
16	Tampax	tampax1	PNG	12.7 KB	139x32

Tabla 23: Datos técnicos de los logos de la Tabla 20.

5.2.2. RESULTADOS

Valor real		**Valor detectado**	
Tiempo	Anuncio	Tiempo	Anuncio
0:14-0:54	Vodafone		-
0:54-1:14	Schweppes	0:00:53 - 0:01:13	schwe0.PNG
1:14-1:34	Película la seducción		
1:34-1:54	Jazztel	0:01:33 - 0:01:53	jazz0.PNG
1:55-2:15	BBVA	0:01:54 - 0:02:14	bbva1.PNG
2:15-2:25	Promocion a3: serie		
2:26-2:36	Once	0:02:24 - 0:02:34	once1.PNG
2:36-2:46	Pelicula		
2:46-2:56	Galicia		
2:56-3:09	Dulcolaxo	0:02:55 - 0:03:08	dulcolaxo1.PNG
3:09-3:29	Mutua madrileña	0:03:08 - 0:03:28	mutua0.jpg
3:30-3:53	Fortasec		
3:53-4:13	Fontbella	0:03:52 - 0:04:12	font0.png
4:13-4:23	Burger King	0:04:12 - 0:04:22	Burger12.PNG
4:23-4:53	Trivago	0:04:22 - 0:04:52	trivago20.PNG
4:55-5:14	Tampax	0:04:52 - 0:05:13	tampax1.PNG
5:14-5:37	Dormidina	0:05:13 - 0:05:36	dormidina-logo3.png
5:37-5:47	Pantene	0:05:36 - 0:05:46	pantene1.PNG
5:47-6:17	Carglass	0:05:46 - 0:06:16	carglass2.PNG
6:17-6:38	Arriaga	0:06:16 - 0:06:36	arriaga.jpg
6:38-6:58	Nocilla	0:06:37 - 0:06:57	nocilla02.png
6:58-7:12	Dulcolaxo	0:06:57 - 0:07:11	dulcolaxo1.PNG

Tabla 24: resultado del test. Los señalados en verde son los acertados. En rojo: erróneos o no detectados. En blanco, partes del vídeo que no se han tenido en cuenta o anuncios para los que no se han encontrado logos en Internet.

Se detectan 15 de los 16 logos del test de forma correcta. Para el logo correspondiente a Vodafone se detectan algunos matches, pero no superan el umbral determinado por las imágenes de entrenamiento. Se descarta, ya que modificar el umbral podría suponer errores en

los otro logos. Por lo que la **probabilidad de acierto es: 93% aproximadamente,** aunque el conjunto de muestras es reducido.

5.2.3. TIEMPOS DE EJECUCIÓN

En paralelo:
Para 8 imágenes: se analizan **0.4811 segundos de vídeo/segundo.**

Secuencial:
Para 10 imágenes: se analizan **0.261 segundos de vídeo/segundo.**

Es decir, en paralelo se analizan 0.4811*25 = 12 fotogramas por segundo, y en secuencial 6.525. Por tanto, la **ejecución en paralelo es casi dos veces más rápida**. La diferencia entre los tiempos de ejecución en paralelo y secuencia es mayor si se analizan más fotogramas por segundo.

Para SURF los resultados y el tiempo de ejecución son similares. No se aprecia una velocidad mayor como se indicaba en la teoría.

6. CONCLUSIONES Y TRABAJOS FUTUROS

6.1 RESULTADOS OBTENIDOS

La segmentación se realiza sin errores. Ésta viene determinada por la aparición de fotogramas en negro, los cuales aparecen durante 2-4 fotogramas y se detectan contando el número de píxeles que no son negros en una imagen. Este umbral se ha determinado en las muestras de entrenamiento y es válido para las muestras de test.

Se obtienen mejores resultados con SIFT que con SURF, para los parámetros establecidos a partir de las muestras de entrenamiento. El tiempo de ejecución es similar para unos 6 minutos de vídeo y 10 muestras. El uso de diferentes *Matchers* no afecta casi nada al resultado.

Respecto al número de fotogramas a analizar por segundo, se ha comprobado que analizar un fotograma por segundo tiene un rendimiento muy similar a analizar más fotogramas en el mismo tiempo. El tiempo de exposición de un logotipo en primer plano es de unos 1-2 segundos por lo general. Por lo que aumentar el número de fotogramas analizados por segundo, reduciría la velocidad, pero produciría más errores (pérdida).

A pesar de que teóricamente estos anuncios identifican bien las esquinas, existen imágenes que pueden dar *matches* en zonas donde no deberían. Dos imágenes de logotipos aparentemente iguales pueden dar resultados diferentes. Por tanto, se debería probar un conjunto de imágenes para cada empresa y escoger la imagen que dé menor probabilidad de error. Para corregir este error se ha utilizado RANSAC, que filtra los resultados correctos, aunque restringe aún más las condiciones para que una imagen sea detectada. Se puede dar el caso de que la imagen supere el primer umbral (el número de matches mínimo que se espera con SIFT o SURF), pero no supere el segundo umbral, determinado por RANSAC. En estos casos, se debe atender al coste del error: es preferible no detectar un anuncio, provocando un error, que arriesgarse a darlo como detectado y provocar más errores en otras secciones del vídeo.

En las pruebas realizadas, se descubre que imágenes que daban buenos resultados en un algoritmo, no pueden valer del todo en otros algoritmos. Se han mostrado dos ejemplos: en una ocasión, el logo no era detectado, o se producían muy pocos *matches* como para considerarlo; el ejemplo opuesto, es la imagen que es detectada erróneamente a lo largo de la secuencia de anuncios.

A la hora de seleccionar imágenes, hay que comprobar que se produce un pico de *matches* en el momento de la aparición del logo en la pantalla. Este deberá superar un umbral predefinido. En el resto de *matches*, este número deberá tener un valor cercano a cero. No hay forma de predecir con rotundidad se un nuevo anuncio puede provocar coincidencias erróneas, pero si no se detectan a lo largo de una prueba de unos 3 ó 5 minutos, y este número permanece cercano a cero, se puede considerar que la imagen es buena, y que la probabilidad de error con pruebas futuras en baja.

Las razones por las que un logo no se detecta con claridad pueden ser varias. En las pruebas se han observado diferentes causas:

- **Logos poco compactos**. Son aquellos, por lo general pertenecientes a alguna promoción temporal, que consisten en una frase, con una fuente de letra fina, y que aparece sobre fondos complejos en movimiento.
- **Logos muy redondeados**, con pocas esquinas claras, también pueden dar problemas.

- **Logos pequeños**. A pesar de que SIFT y SURF abordan el problema de la escalabilidad, se ha observado que es más difícil detectar logos que se muestran pequeños en una parte de la imagen. Las esquinas son menos evidentes en estos logos teniendo en cuenta que la resolución no cambia.
- **Aparición breve de la imagen en pantalla**. En la opción por defecto del algoritmo es analizar 1 fotograma (de los 25 fps) cada segundo. Si el logo permanece quieto en un tiempo inferior, su detección será peor.
- **Imagen del logotipo afectado por fuentes externas**. Cuando el logo se ve afectado por algún efecto de iluminación, brillo, cambios en el color, ruido. etc. Las diferencias con el logo original son mayores, y el número de coincidencias disminuye.

El uso de procesos paralelos reduce la velocidad significativamente al utilizar más capacidad de la CPU.

6.2. VENTAJAS E INCONVENIENTES

6.2.1. VENTAJAS

- Solo es necesaria **una muestra** para poder comparar con la imagen principal (no entrenamiento como en otros algoritmos de aprendizaje automático).
- **Implementación fácil.** Solo requiere tres funciones de OpenCV (creación de descriptores, matchers, y algoritmos como RANSAC) y algunas para modificar parámetros de estos algoritmos.
- **Fácil identificación del problema.** En la sección de las pruebas con el conjunto de entrenamiento se ha representado gráficas con el número de *matches* respecto al tiempo. Mediante los documentos generados como salida y su representación gráfica, se puede identificar fácilmente cuales son las muestras que dan problemas, y en qué magnitud y tiempo.

6.2.2. INCONVENIENTES

- **La muestra tiene que ser lo más parecida a lo esperado**. En el caso de un logotipo, deberá tener el color, la forma, y a ser posible el fondo característico. Si la muestra tuviese un fondo distinto, lo cual es algo habitual, puede producir errores.
- **La determinación de los parámetros de los algoritmos se hace de forma manual**. Estos parámetros pueden ser: la distancia entre el mejor resultado y el segundo, el número de árboles y el umbral de *matches*. Estos se han ido modificando según los resultados de las muestras de entrenamiento, midiendo se rendimiento. En algoritmos de aprendizaje máquina este error se mide automáticamente, y con ellos, los parámetros que lo producen.
- **La selección de imágenes debe ser cuidadosa**. En este proyecto, se asocia una sola imagen con un solo anuncio para no crear redundancia. Esta imagen debe dar los mejores resultados para ser válida. En algoritmos de aprendizaje automático, el conjunto de imágenes destinado a la detección (acierto), y el de no detección (fallo), contiene gran cantidad de archivos y las imágenes pueden contener algunas imperfecciones. Aquellas imágenes que un ser humano detecto como válidas en la vida real, deberían ser válidas para la máquina.

6.3. TRABAJOS FUTUROS

6.3.1. POSIBLES MEJORAS

Para este proceso se han utilizado *Feature Matching,* pero existen otras alternativas. Se ha escogido este método debido a que los logotipos no suelen sufrir cambios de color, forma u orientación significativos. Para justificar el uso de un algoritmo de **aprendizaje máquina**, serían necesarias muchas muestras, con fondos diferentes.

Utilizar **Haar Cascade** es un algoritmo interesante. Básicamente crea un fichero que almacena todas las características que hacen identificable una imagen, después de un proceso de aprendizaje. Éste es muy utilizado por ejemplo en el reconocimiento de una cara de una persona cuando se va a realizar una fotografía con un actual.

El programa es **escalable**. Si se introduce otro vídeo como entrada detectará los mismos anuncios, de los cuales se tengan los logotipos en una carpeta. Una funcionalidad adicional sería que se mostrase un vídeo, y una persona introdujese el nombre de la empresa anunciada en el programa. De esta forma ser iría entrenando a la máquina, y formando un modelo de decisión para poder detectar anuncios en un futuro.

En cuanto a la **velocidad**, el algoritmo está optimizado para utilizar mucha capacidad de procesamiento de un ordenador, debido a la creación de hilos, por lo que para conseguir que más imágenes se analicen simultáneamente, se puede utilizar un ordenador más potente, por ejemplo, contratando un servidor externo.

El programa únicamente trabaja con imágenes, pues solo analiza la intensidad de los píxeles. Pero se pueden tener otro tipo de datos que puede dar pistas: el **audio**. Reconocer el nombre de la marca, una sintonía, o el sonido de ambiente, puede ayudar a clasificar mejor el anuncio.

BIBLIOGRAFÍA

[Bay06] BAY, Herbert, TUYTELAARS, Tinne y VAN GOOL, Luc. (2006). *SURF: Speeded Up Robust* Features.

[Bradski08] BRADSKI, Gary,.KAEHLER y Adrian (2008) *Learning OpenCV.* O'Reilly

[Calonder10] CALONDER, Michael, LEPETIT, Vincent, STRECHA, Christoph y FUA, Pascal (2010). *BRIEF: Binary Robust Independent Elementary Features.* 11th European Conference on Computer Vision (ECCV), Heraklion, Crete. LNCS Springer.

[Gladwell00] GLADWELL, Malcolm *(2000). La clave del éxito.*

[Harris88] HARRIS, Chris y STEPHENS, Mike. (1988). *A Combiner Corner and Edge Detector.*

[Kahneman11] KAHNEMAN. (2011). Daniel, Pensar Rápido, Pensar Despacio.

[Lowe04] LOWE, David G. (2004) *Distinctive Image Features from Scale-Invariant Keypoints.*

[Lutz13] LUTZ, Mark (2013). *Learning Python.* O'Reilly.

[Muja09] MUJA, Marius y LOWE, David G. (2009) *Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration.*

[Rosten06] ROSTEN, Edward y DRUMMOND, Tom. (2006) *Machine Learning for High-Speed Corner Detection.*

[Rublee11] RUBLEE, Ethan, RABAUD, Vincent, KUNOLIGE, Kurt y BRADSKI, Gary (2011). *ORB: an efficient alternative to SIFT or SURF*. Proceedings / IEEE International Conference on Computer Vision. IEEE International Conference on Computer Vision

[Severance16] SEVERANCE, Charles Russell (2016). *Python for Everybody: Exploring Data in Python 3*

[Shi94] SHI, Jianbo y TOMASI, Carlo. (1994). *Good Features to Track.*

[Solem12] SOLEM, Jan Erik (2012). *Programming Computer Vision* O'Reilly.

[Taleb07] TALEB, *Nassim Nicholas,* (2007) *. El Cisne Negro: el impacto de lo altamente improbable.*

RECURSOS

Git Hub: código libre de ejemplos. También útil para subir el código propio, aunque la versión privada cuesta dinero. https://github.com/

Kaggle: página con algunos tutoriales, y recursos como imágenes de entrenamiento. https://www.kaggle.com/

Librería de Python 3: https://docs.python.org/3/library/index.html

Source Forge: repositorio con algunos recursos de software. https://sourceforge.net/

Stack Over Flow: foro para dudas de programación. https://stackoverflow.com/

ANEXO: MANUAL DE USUARIO

A EJECUCIÓN DEL PROGRAMA

Pasos a seguir para la ejecución del programa:

A.1. PASOS PREVIOS

1. Colocar el programa .py en un directorio
2. Crear una carpeta llamada «datosX» que contenga las imágenes de los logotipos que se quieran buscar.
3. Crear una carpeta llama «datos» que contenga los vídeos a analizar.
4. Poner en el directorio del programa el archivo de vídeo que se quiera analizar (secuencial o paralelo)

A.2. EJECUCIÓN

1. Ejecutar el archivo .py del proyecto.
   Para ello, situarse en el directorio del proyecto donde se encuentre el programa .py, y ejecutar la siguiente línea de comando:

   ```
   python programa.py datosX video_test1.ts
   ```

   Un ejemplo de ejecución es el siguiente:

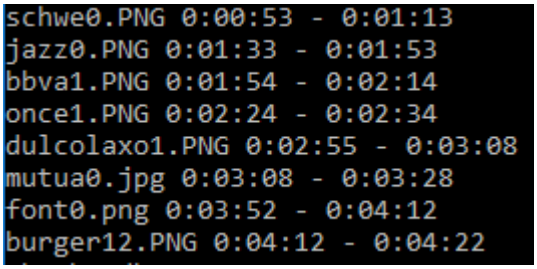
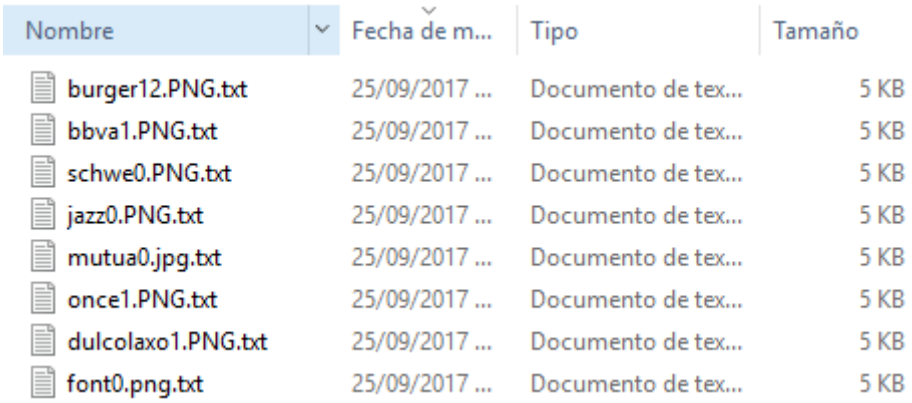


Figura 49: Ejemplo de ejecución por consola.

2. También se crean dos carpetas en el directorio.

- reg: contiene el registro de cada logotipo a buscar en formato .txt. Este registro contiene los *matches* que se han producido para la última ejecución que se ha realizado del programa, en los tiempos especificado. Estos pueden ser tres: 25, 5 o 1 fotograma por segundo.

- Registro_final: contiene los registros como en la carpeta reg, pero estos siempre guardarán los registros de cada segundo. Para ello se tomarán los *matches* de cada segundo (5 en el caso de ser 25 fotogramas/segundo, y 25 en el caso de 25 fotogramas/segundo) y se hará la media aritmética.



Nombre	Fecha de m...	Tipo	Tamaño
burger12.PNG.txt	25/09/2017 ...	Documento de tex...	5 KB
bbva1.PNG.txt	25/09/2017 ...	Documento de tex...	5 KB
schwe0.PNG.txt	25/09/2017 ...	Documento de tex...	5 KB
jazz0.PNG.txt	25/09/2017 ...	Documento de tex...	5 KB
mutua0.jpg.txt	25/09/2017 ...	Documento de tex...	5 KB
once1.PNG.txt	25/09/2017 ...	Documento de tex...	5 KB
dulcolaxo1.PNG.txt	25/09/2017 ...	Documento de tex...	5 KB
font0.png.txt	25/09/2017 ...	Documento de tex...	5 KB

Figura 50: Ejemplo de lista de registros en una carpeta después de ejecutar el programa.

3. El programa ejecutará varios hilos a la vez, y mostrará por pantalla la ejecución mientras dure el vídeo.